Deadline Miss Models for Temporarily Overloaded Systems

Deadline Miss Models for Temporarily Overloaded Systems

Von der Fakultät für Elektrotechnik, Informationstechnik, Physik

der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines Doktors

der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von Zain Alabedin Haj Hammadeh
aus Aleppo, Syrien

eingereicht am: 29.01.2019

mündliche Prüfung am: 28.05.2019

1. Referent: Prof. Dr.-Ing. Rolf Ernst
2. Referent: Prof. Marco Di Natale
3. Referentin: Dr. Sophie Quinton
4. Referent: Apl. Prof. Dr.-Ing. Wael Adi (Vorsitzender)

Druckjahr: 2019

For Raghad and Ayham

ABSTRACT

A wide range of embedded systems falls into the category of safety-critical systems. Such systems impose different levels of safety requirements depending on how critical the functions assigned to the system are and on how humans interact with the system. Safety requirements involve timing constraints, the violation of which may lead to a system failure. Timing constraints are graded from soft to hard real-time constraints. While satisfying soft real-time constraints requires only best-efforts guarantees, hard real-time constraints are best-treated with worst-case analysis methods for verifying all timing constraints. Weakly-hard real-time systems have extra demands on the timing verification as they tolerate few deadline-misses in certain distributions. Applying worst-case analysis methods, in which a task is schedulable only when it can meet its deadline in the worst-case, to weakly-hard real-time systems questions the expressiveness of the computed guarantees. Considering tolerable deadline-misses raises the need for weakly-hard schedulability analyses to verify weakly-hard real-time constraints and to provide more expressive guarantees.

This thesis addresses the schedulability analysis problem of weakly-hard real-time systems. It presents an efficient analysis to compute weakly-hard real-time guarantees in the form of a deadline miss model for various system models.

The first contribution is a deadline miss model for a temporarily overloaded uniprocessor system with independent tasks under the Fixed Priority Preemptive and NonPreemptive scheduling policy (FPP & FPNP) using Typical Worst-Case Analysis. In our application context, the transient overload is due to sporadic tasks, for example, interrupt service routines. We adopt the proposed analysis to compute deadline miss models for independent tasks under the Earliest Deadline First (EDF) and Weighted Round-Robin (WRR) scheduling policies.

In the second contribution, we extend the analysis to compute deadline miss models for task chains. The extension is motivated by an industrial case study.

The third contribution of this thesis targets the system extensibility to budget under-specified tasks in a weakly-hard real-time system. Adding recovery or reconfiguration tasks such that the system still meets its weakly-hard timing con-

straints is of interest of an industrial case study (satellite on-board software) that is considered in this thesis.

We show formally and in experiments with synthetic as well as industrial test cases that the analysis presented in this thesis can consider various scheduling policies (FPP, FPNP, EDF, WRR), and can be extended to cover both independent and dependent tasks. The thesis provides two practical solutions for two industrial case studies, which are involved exclusively in a collaboration project between Thales Research & Technology and iTUBS, which is a technology transfer company associated with Technische Universität Braunschweig. The results are thus of real practical value to be considered in the design process of weakly-hard real-time systems.

KURZFASSUNG

Eine große Zahl von eingebetteten Systemen fällt in die Kategorie der sicherheitskritischen Systeme. Solche Systeme stellen unterschiedliche Sicherheitsanforderungen, je nachdem wie kritisch die dem System zugewiesenen Funktionen sind und wie Menschen mit dem System interagieren. Sicherheitsanforderungen beschreiben insbesondere das geforderte Echtzeitverhalten, dessen Verletzung zu einem Systemausfall führen kann. Anforderungen an das Echtzeitverhalten können unterschiedlich strikte Echtzeitbedingungen umfassen. Während die Erfüllung weicher Echtzeitbedingungen nur wenn möglich gefordert ist, braucht es zur Gewährleistung harter Echtzeitbedingungen die Anwendung von Worst-Case-Analysen zur Überprüfung der zeitlichen Bedingungen in allen Fällen. Die besondere Kategorie der schwach-harten Echtzeitsysteme hat zusätzliche Anforderungen an die Timing-Verifikation, da sie wenige Deadline-Überschreitungen mit bestimmten Mustern tolerieren. Das stellt die Aussagekraft der mit Methoden der Antwortzeitanalyse berechneten Grenzen in Frage und erhöht den Bedarf an Analysen, um die schwach-harten Echtzeitbeschränkungen zu verifizieren und aussagekräftigere Garantien zu liefern.

Diese Arbeit befasst sich mit dem Problem der Scheduling-Analyse von schwach-harten Echtzeitsystemen. Es stellt eine effiziente Analyse zur Berechnung von schwach-harten Echtzeitgarantien in Form eines Deadline Miss Modells für verschiedene Systemmodelle dar.

Der erste Beitrag ist ein Deadline-Überschreitungsmodell für ein temporär überlastetes Uniprozess-System mit eigenständigen Aufgaben im Rahmen der Fixed Priority Scheduling Policy (FPP & FPNP) mittels Typical Worst-Case Analysis. In unserem Anwendungskontext sind sporadische Tasks die Ursache von temporärer Überlast, wie zum Beispiel Interrupt Service Routines. Wir adaptieren die vorgeschlagene Analyse für Earliest Deadline First (EDF) und gewichtete Round-Robin (WRR) Scheduling.

Zweitens erweitern wir die Analyse, um ein Deadline-Überschreitungsmodell für Taskketten zu berechnen. Die Erweiterung wird durch eine industrielle Fallstudie motiviert.

Der dritte Beitrag dieser Arbeit zielt auf die Erweiterbarkeit des Systems bei unterdefinierte Tasks in einem schwach-harten Echtzeitsystem ab. Ziel ist es, ein Budget für Wiederherstellungs- oder Rekonfigurationstasks herzuleiten, so dass das System immer noch seine schwach-harten Echtzeitbedingungen erfüllt. Dies ist von Interesse für eine industrielle Fallstudie (Satellitensoftware), die in dieser Arbeit berücksichtigt wird.

Wir zeigen formal und in Experimenten mit synthetischen sowie industriellen Daten, dass die in dieser Arbeit vorgestellte Analyse von hoher Flexibilität ist, um verschiedene Planungsstrategien (FPP, FPNP, EDF, WRR) zu berücksichtigen und um sowohl unabhängige als auch abhängige Tasks zu erweitern werden kann. Die Arbeit zeigt zwei praktische Lösungen für zwei industrielle Fallstudien, die in einem Kooperationsprojekt zwischen Thales Research & Technology und iTUBS, einem Technologietransferunternehmen der Technischen Universität Braunschweig, erarbeitet wurden. Die Ergebnisse sind somit von echtem Praxiswert, und können im Designprozess schwach-harter Echtzeitsysteme als Hilfestellung dienen.

Contents

1 INTRODUCTION

"It is all about timing."

— Someone

Real-time systems are computer systems that must satisfy *timing constraints* , i.e., respond before a predefined time interval elapses. The timing constraints are imposed by the system environment or operational requirements [112]. Real-time systems exist in various applications, for instance in automotive, e.g., airbags, braking systems and electronic fuel injection; in space, e.g., the platform of satellites [52]; and in defense, e.g., aerial video tracking system [54]. Figure 1.1 illustrates a mission critical real-time system: aerial video tracking system where the system is dedicated to displaying high-quality video images and detecting and tracking moving objects. Both comprise functions with timing constraints. For instance, the high-quality video runs at $25f/sec$ (frame per second) which implies that the video frame processing subsystem has to deliver a processed frame every $40ms$ to the monitor.

Violating system timing constraints may jeopardize the correctness of the delivered service which leads to a system *failure*, e.g., an airbag in a vehicle has to respond within a specific time interval, as otherwise the passenger's life may be in danger. Therefore, verifying the system performance against timing constraints is of high importance particularly for *safety-critical* systems, in which a failure may lead to severe damage to people or property.

Today's trend in designing real-time systems (particularly in the automotive domain) is to comprise networked functions realized by interactive Distributed Real-Time Systems (DRTSs) [104], which leads to a better utility of the available resources. Figure 1.2 shows the growing of number of Electronic Control Units (ECUs), buses and signals in the Mercedes-Benz E-Class through five generations. The fast-growing of network communication comparing to the number of ECUs reflects the high degree of integration on each ECU. Although not all these signals belong to safety-critical functions, designing such complex systems is a serious challenge in the context of verifying the system performance against the timing

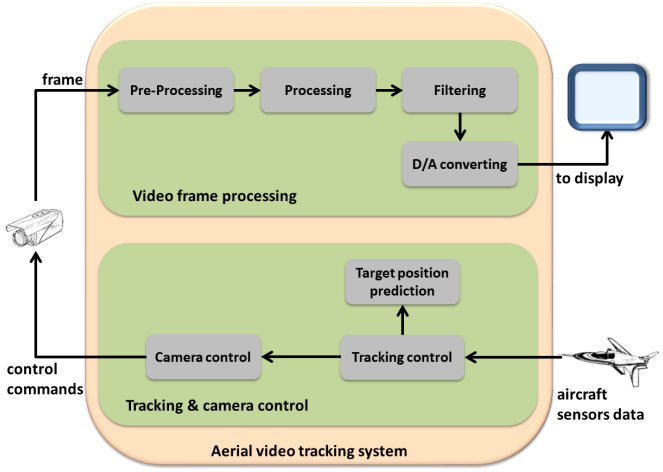

Figure 1.1: The functional view of an aerial video system, based on [54]. The video frame procesing subsystem is dedicated to displaying high-quality video images, while the Tracking & camera control subsystem is dedicated to detecting and tracking moving objects.

constraints. To give an intuition on how complex the analysis can be, let us consider the architectural view of the aerial video system example in Figure 1.3. A heterogeneous architecture (FPGA, GPP, GPU) realizes the functions leading to complex scenarios of interdependencies, which impact the system performance dramatically [59]. Verifying the system performance against the timing constraints requires therefore a sophisticated performance analysis that comprehensively captures system properties and function dependencies. The next section is dedicated to survey the main classes of performance analysis.

1.1 Performance Analysis

Consider GPP_1 in Figure 1.3, and let us assume that each of the four functions is executed by a single *task*, which is an executable entity of work [111]. Each task needs to occupy GPP_1 for an amount of time to be executed; this time is called the *execution time*. Because of the interdependencies, each task is a subject to be blocked, delayed, or preempted by other tasks. The time that a task need to complete its execution including the execution time, blocking, delay and preemption is called the *response time*. A *deadline* is the timing constraint on a task completion due time [18]. Considering the timing properties of tasks in tracking & camera control subsystem, the subsystem, which is a chain of tasks, has an *end-to-end delay* in delivering control commands to the camera.

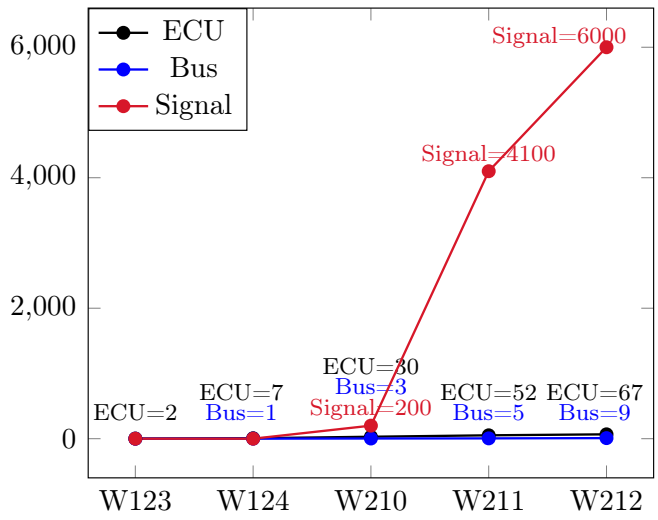

Figure 1.2: Example: the growing of number of ECUs, buses and signals in the Mercedes-Benz E-Class through five generations, based on [14].

Verifying the system performance requires computing guarantees on the timing properties such as execution times, response times and end-to-end delays. Four main classes of performance analysis exist to estimate the performance of real-time systems, namely testing, simulation-based analysis, probabilistic analysis, and worst-case analysis.

1.1.1 Testing

Testing is possible if the functions are complete because it applies external test patterns on an executable model. Testing is generally satisfactory as it shows how the system behaves in reality. However, testing does not guarantee a full coverage of system states unless the test patterns have been proven to cover all possible states. Moreover, testing is costly as it is applied only to the final design, and with the increasing of the system size and the behavior complexity it is tough to define test patterns that cover all system states [79].

1.1.2 Simulation-based analysis

Simulation-based performance verification [13, 56] is prevalent among the industry, see, e.g., [76, 72]. In simulation-based methods, abstract models of the system and the physical environment are coupled to capture dynamic and complex interactions. Unlike functional verification, timing verification requires complex timed models rather than the simplified un-timed models [99]. Simulation-based tools

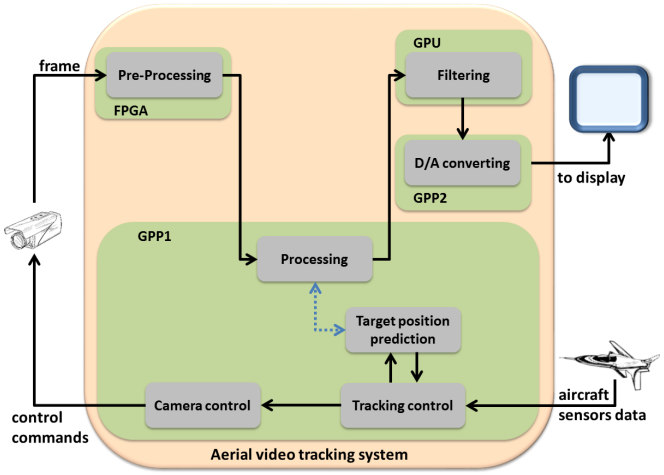

Figure 1.3: The architectural view of an aerial video system, based on [54]. The functions are mapped to four different processors. The blue dotted arrow indicates an interdependency via a shared resource.

have to be fed with stimuli sequences in order to cover all corner-cases to prevent system failures. Identification of comprehensive simulation stimuli is practically not realistic for modern complex systems. Instead, simulations like Monte Carlo select random input to compute the output of the model. Typically, Monte Carlo simulation repeats the computation hundreds or thousands of times based on different randomly-selected inputs [101]. The difficulty of identifying full corner-cases coverage stimuli disputes the eligibility of these methods to provide guaranteed bounds for real-time systems, particularly the safe-critical ones. However, simulation provides estimates of the average system performance [122].

1.1.3 Probabilistic analysis

Probabilistic analyses consider a real-time system model in which at least one parameter, e.g., execution time, follows a probabilistic distribution. They (probabilistic analyses) return probabilistic bounds on the system performance, for instance the response time of a task is below a certain threshold with the probability $\mathbb{P}_{rt}$. Note that the term *stochastic* is often used rather than probabilistic, because random variables describing, e.g., the execution times of instances of a given task can be seen as a stochastic process.

The analysis method is based on Markov process modeling, which reasons probabilistically about the steady-state behavior of the system. It provides both analytical and numerical solutions for the *deadline-miss* probabilities of tasks, i.e., the probability of having the response time $>$ the deadline, by computing the complete

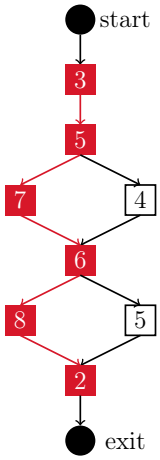

Figure 1.4: The longest path of a CFG where each box represents a sequential computation and the number in the box represents the time needed to execute this box, based on [124].

probability function of the response time of each task [31].

Probabilistic bounds can be applied for modeling the impact of error occurrences on the response time of fault-tolerant systems, since faults can usually be approximated as randomly distributed [6], and for *soft real-time systems* [18], i.e., missing deadlines does not jeopardize the system correctness. Probabilistic bounds are sufficient to guarantee an acceptable level of Quality-of-Service (QoS), e.g., multimedia system in automotive.

The fundamental challenge of probabilistic analysis in timing verification is twofold: 1) its high computational complexity, and 2) representing the execution time of task instances with independent random variables [90].

Currently, most approaches [31, 71, 20, 21] are for single-processor systems and cannot be extended to distributed systems because of the methods they use. A notable exception is probabilistic real-time calculus (RTC) [103], which follows the Compositional Performance Analysis (CPA), therefore, it is applicable to DRTSs.

1.1.4 Worst-case analysis

Worst-case analysis addresses defining the worst-case and best-case behaviors of a system, it guarantees full performance corner-case coverage. There are plenty of worst-case analysis based methods as they got the attention when the simulation-based methods became ineffective to capture the corner cases of complex systems [100, 124].

Worst-case methods compute upper and lower bounds on the timing properties

of a system depending on well-defined models, e.g., arrival curves [99]. Consider the task "Processing" in GPP_1 in Figure 1.3, the worst-case analysis computes the *worst-case execution time* of the task, i.e., the longest time that the task needs to execute the longest path in its Control Flow Graph (CFG). Figure 1.4 illustrates the longest path of a CFG in which each box represents a sequential computation, and the number in the box represents the time needed to execute this box. To compute the *worst-case response time* of the task, i.e., the longest response time, the worst-case analysis takes into account the worst-case execution time, the maximum blocking, the maximum delay and the maximum preemption.

The worst-case scenario is over-approximated because the complex dependencies are conservatively taken into account. Therefore, worst-case methods may report too pessimistic bounds. Actually, the main challenge facing worst-case methods is the pessimism of the computed bounds. However, considering the worst-case scenario limits the need to check each individual system state which makes worst-case methods ideal for design-space exploration.

Figure 1.5 illustrates the computed worst-case execution time of a task returned by three classes of performance analysis (simulation, probabilistic, worst-case). The exact worst-case execution time is a reference. On the one hand, simulation reports an execution time smaller than the exact one, which makes it not preferable for safety-critical systems. On the other hand, worst-case analysis reports an over-approximated bound but yet safe. Probabilistic analysis returns the probability of exceeding the execution time of a certain threshold. Consequently, the worst-case analysis is preferable for safety-critical systems, for which, the analysis has to cover all corner cases. However, the next section shows that many real-time systems, including some safety-critical systems, need a *less-than-worst-case* analysis, which is not listed in this section.

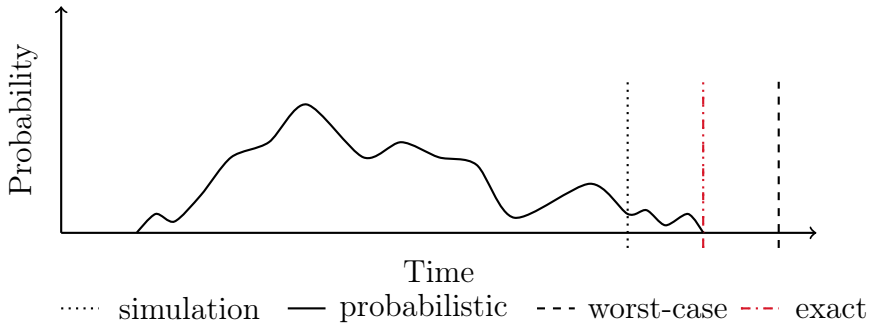

Figure 1.5: Worst case execution time by simulation, probabilistic and worst-case analyses, based on [90].

1.2 Beyond Worst-Case Timing Analysis

In real-time systems, missing a deadline is bound to have consequences. A real-time task is considered *hard* if missing its deadline may jeopardize the system correctness. If missing the deadline causes a performance degradation but no damage to the system, then the task is considered *firm* [18]. If missing the deadline causes no damage and the task result is still useful for the system, then the task is considered *soft*. Safety-critical tasks are considered in this context to be hard real-time tasks. Because of the critical mission that the aerial video tracking system has, Figure 1.3, all tasks are considered to be hard real-time tasks. However, the task "Processing", which belongs to the video frame processing subsystem in Figure 1.1, can miss few deadlines without jeopardizing the correctness of the subsystem due to the following fact: The human brain can detect and track an object using 13 frame per second and any video runs faster than $20f/sec$ makes no more improvement [16]. Because each task in the subsystem rather than "Processing" is mapped to its own resource, we can assume that the task "Processing" is prone to deadline-misses, which may lead to end-to-end deadline-miss. As the subsystem provides a $25f/sec$ video, the task "Processing" may miss up to 5 deadlines every 1 second without jeopardizing the correctness of the subsystem. Therefore, the task "Processing" does not fit in any of the three categories: hard, firm and soft.

When missing a deadline is admissible, the distribution of the deadline-misses have to be bounded. Worst-case analyses are strict in verifying a real-time task, and they do not bound such a distribution, rather, they tell us whether a real-time task meets its deadline all the time or not. Probabilistic analysis, as has been shown in the last section, can describe the distribution of deadline-misses as a probability [31]. However, information about the sequence of occurrence is missing in probabilistic bounds. Such information has quite an impact on the system performance [85]. To clarify this point, suppose that the aerial video tracking system in our example was designed in such a way that the task "Processing" may miss its deadline. Assume now that a probabilistic analysis reports that "Processing" has the probability of $P_{miss} = 0.07$ to miss its deadline, which means that "Processing" may miss 7 deadlines in a sequence of 100 frames (or 7 deadline-misses every 4 seconds) with no further information about the distribution of these deadline-misses. On the one hand, if these 7 deadline-misses occur in a row, the system may fail in delivering a correct service because it is more than the admissible bound (5 deadline-misses every 1 second). On the other hand, if they have the distribution of, for instance, 5 deadline-miss followed with 20 deadline-hits[1], the correctness of the subsystem will not be impacted.

[1] In this thesis, we use "deadline-hit" (noun) as the opposite of "deadline-miss" and therefore the plural form is "deadline-hits".

As a conclusion, neither worst-case analysis nor probabilistic analysis are suitable for tasks for which missing deadlines is admissible and there is a need for a less-than-worst-case analysis. Such an analysis is motivated with the fact that admitting deadline-misses is of a high significance to relax and/or optimize the design of real-time systems [83], for instance, optimizing cost/performance in control systems utilize the admissible deadline-misses [23]. To exploit a deadline-miss in relaxing and optimizing the design of real-time systems, few questions have to be answered: Where does the deadline come from? What does a deadline-miss mean and which consequences does it have? The next two subsections try to answer these questions.

1.2.1 Deadline in the design process

Deadlines, generally speaking, reflect the underlying physical nature of the system [63]. For example, in reactive systems a task has to complete before the next instance is activated (deadline = period).

Safety-based and performance-based criteria are used to establish deadlines [73], thus, a deadline has different indications, e.g., [35]

- it constrains function delays in order to adjust the system timing;
- it avoids data loss due to overwriting or early reading, which guarantees the data consistency;
- it avoids stack overflow due to overlapping job executions, which protects the memory usage.

In our example, Figure 1.3, deadlines are for system timing.

1.2.2 Intrinsically deadline-miss-tolerant real-time systems

Depending on the significance of a deadline, the consequences of a deadline-miss can be predicted. Each deadline-miss is an error, which may cause a system failure [4]. A task *tolerates* a deadline-miss when this deadline-miss causes no failure without error handling. The key point here is that *not every deadline-miss causes a failure*. Control systems can intrinsically tolerate a bounded number of deadline-misses due to the robustness of control algorithms [85]. Another example of intrinsically deadline-miss-tolerant systems is the animated displays in which frames must be updated at a certain rate to provide continuous motion; in this case the acceptance of deadline-misses is related to human perception as we saw for the subsystem video frame processing in our example. In the subsystem tracking &

camera control, the acceptance of deadline-misses is determined by the robustness of the smart algorithm in the "Target position prediction" task.

To conclude, we say that the hard real-time model cannot fully describe tasks that tolerate few deadline-misses and the worst-case analysis is too strict to verify the timing of such tasks. There is actually a need for a less-than-hard model and a less-than-worst-case analysis. *Weakly-hard real-time systems* [7] have been defined to model such systems.

1.3 Weakly-hard real-time systems

A weakly-hard real-time system has been defined for the first time in [7] as

"A system in which the distribution of its met and missed deadlines during a window of time w is precisely bounded."

That defines a new category of real-time tasks, namely weakly-hard. The notation (m, k) is used to represent the distribution of deadline-hits and deadline-misses. This notation originates from the work on (m, k)-firm systems [47], which addresses the same type of systems. A task is said to be weakly-hard if it can tolerate m deadline-misses in a sequence of k deadlines. Formal definitions and discussions take place in Chapter 2. The task "Processing" in our example can tolerate $m = 5$ in a sequence of $k = 25$ depending on the fact presented in Section 1.2.

Clearly, the worst-case analysis cannot provide guarantees for weakly-hard real-time tasks, thus, Bernat et al. proposed an analysis in [7] to compute the distribution of deadline-misses for real-time periodic tasks with fixed offset. They computed the number of deadline-misses in a time-window of k consecutive jobs along the *Hyper-period*, i.e., the least common multiple (lcm) of all task periods. Sun and Di Natale also proposed an analysis in [113] to compute tight bounds for offset-free periodic tasks. The authors proposed an Mixed-Integer Linear Programming MILP to check all possible scenarios within a time-window of k consecutive jobs. Therefore, [113] can provide tighter bounds than [7] but with higher complexity.

This thesis considers weakly-hard real-time tasks and proposes an analysis to compute weakly-hard real-time bounds for variety of system models as will be elaborated in this thesis. The contribution of this thesis is addressed and detailed in the next section.

1.4 Research Objectives and Contribution

Throughout the introduction of this thesis, the necessity of tight and safe bounds on the timing properties of real-time systems has been elaborated. This doctoral thesis aims to bring more attention to the modeling and analysis of weakly-hard real-time systems. Leveraging the properties of weakly-hard real-time systems leads to relax the traditional hard real-time constraints and to compute more expressive, tight and yet safe bounds for weakly-hard real-time tasks.

The core contribution of this thesis is: computing a deadline miss model for independent weakly-hard real-time tasks that are scheduled on a temporarily overloaded uniprocessor system. A part of this work has been achieved in the course of a collaboration project between Thales Research & Technology (TRT) and iTUBS which is an associate company of Technische Universität Braunschweig (TUBS), responsible for technology transfer. The overall objective of the collaboration project consists in developing a framework of integrated tools assisting the system architect in evaluating the timing behavior of the architecture based on weakly-hard real-time guarantees in the form of a deadline miss model. The collaboration project involved two case studies, which are considered and presented in this thesis.

The contribution can be detailed as follows:

- Computing weakly-hard real-time guarantees in the form of a deadline miss model for independent tasks under Fixed Priority Preemptive and Nonpreemptive (FPP & FPNP) scheduling policies.
- The proposed analysis is adopted to compute deadline miss models for independent tasks under Weighted Round-Robin (WRR) and Earliest Deadline First (EDF) scheduling policies.
- An extension to address the first case study of TRT. The extension computes deadline miss models for weakly-hard real-time systems with task dependencies, that is to say, for a chain of tasks.
- System extensibility to budget recovery and reconfiguration tasks in a weakly-hard real-time system. This extension addresses the second case study of TRT.

The work presented in this thesis is significant to compute weakly-hard real-time guarantees for tasks that can tolerate deadline-misses in a temporary overloaded system. This work is not a stand-alone work, rather, it is a basis of two follow-up dissertations. The first one [2] is pursued at TUBS to provide weakly-hard real-time guarantees in the form of a deadline miss model for a DRTS. The second [38] is pursued at the University of Grenoble Aleps; it aims to bring formal

proofs of the weakly-hard analysis. Both topics build upon the analysis proposed in this thesis.

1.5 Outline

This thesis is organized as follows: Chapter 2 recalls basic definitions and formally presents the system model. The core contribution of this thesis is shown in Chapter 3 where an analysis to compute weakly-hard real-time guarantees in the form of a deadline miss model will be presented for FPP and FPNP scheduling policies. Later, the proposed analysis will be adopted to consider WRR and EDF scheduling policies. In Chapter 4, the applicability, scalability, and pessimism of the proposed deadline miss model will be open to question. The discussion will be supported with experimental results. Two extensions will be motivated and presented in Chapters 5 and 6. In Chapter 5, the extension to a deadline miss model for task chains will be motivated by and presented with an industrial case study provided by TRT. The second extension is in Chapter 6. The system extensibility will be studied for weakly-hard real-time systems to safely add recovery and reconfiguration tasks to the system. This chapter is driven by a second case study provided by TRT as well. Finally, conclusions about this thesis will be drawn in Chapter 7.

2 SYSTEM MODEL AND PROBLEM FORMULATION

"It isn't that they can't see the solution. It's that they can't see the problem."
— G. K. Chesterton

Missing deadlines and weakly-hard real-time systems are intrinsically linked. Missing deadlines are also intrinsically linked with *overload* conditions, in which the processing capacity offered by the resource cannot satisfy the computational capacity requested by the tasks [18]. Overload conditions can be raised by different sources, such as interrupt service routines triggered by sensors or exception handling routines triggered by the operating system. External overload conditions can cause, e.g., higher event density. Cyclic-and-spontaneous messages in a Controller Area Network (CAN) bus can suffer from sporadic overload jobs [89]. In the design of real-time systems, it is not uncommon to fully specify the task related to the main mission of the system and later consider the exception handling routines and the interrupt service routines [52]. The execution of such routines may, however, perturb the execution of main mission tasks, leading to deadline-misses. In other words, those routines make the system overloaded. If the main mission tasks can tolerate deadline-misses, which means they are weakly-hard real-time tasks, a weakly-hard analysis is needed to verify their schedulability under the overload conditions.

This chapter presents the system model considered in this thesis and formally defines the problem addressed in the main contribution. The chapter encompasses formal definitions of the principal concepts that the thesis relies on, e.g., temporarily overloaded system, weakly-hard real-time task and weakly-hard schedulability.

2.1 System Model

This section follows [82, 5] and focuses on the timing model of a real-time system. This section goes further to identify weakly-hard real-time systems. The section

starts with general definitions and moves to more specific definitions to formally recognize the system model. Let us consider a simple sensor-control-actuator sys-

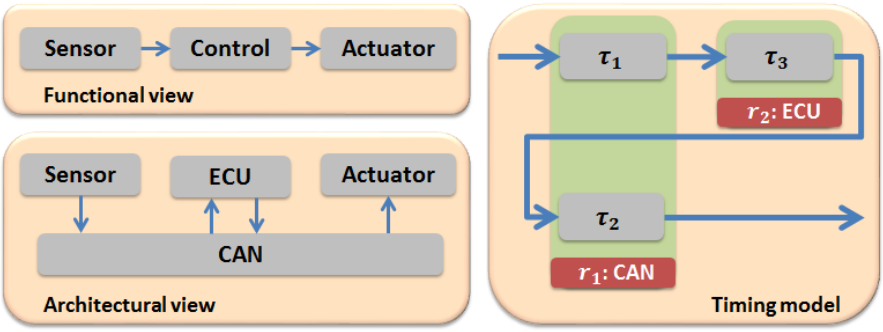

Figure 2.1: The timing model of a simple sensor-control-actuator system.

tem as in the left part of Figure 2.1. The functional view shows that there is one function, the control algorithm, that reads from the sensor and sends control commands to the actuator. This system is realized by an ECU and a CAN bus; the sensor sends the value to the ECU as a message via the CAN bus, in turn the ECU sends the control command as a message to the actuator via the CAN bus. The timing model of this simple system, which is adequate for timing verification, is an abstraction from the functional view and the architectural view. The right part of Figure 2.1 shows the timing model of the simple system.

For the sake of consistency, the basic modeling formalism required for this thesis will be presented next. A list of notations is shown in Table 2.1.

2.1.1 Principal definitions

Definition 2.1 (System). *A system* $\mathbb{S}$ *comprises a set of tasks mapped to a set of resources.*

Definition 2.2 (Resource). *A resource* r *is a physical entity provided with a scheduler, which arbitrates between tasks according to a scheduling policy.*

Definition 2.3 (Task). *A task* τ *is modeled as a tuple* $(C, \mathcal{A}, D)$*, where* C *is the worst-case execution time,* $\mathcal{A}$ *is the activation model and* D *is the relative deadline.*

Definition 2.4 (Execution time). *The time needed to execute the task, including overheads caused by the operating system, i.e., context switching, and taking into account the hardware specifications.*

A task τ_i contends for the resource after being activated by an *event*, i.e., an interrupt or a message reception. The scheduling policy determines when τ_i can

notation	description
$\mathbb{S}$	System
r	Resource
τ	Task
C_i	Worst-case execution time of τ_i
D_i	Relative deadline of τ_i
d_i^n	Absolute deadline of τ_i
R_i^+	Worst-case response time of τ_i
$\aleph_i$	Activation trace for τ_i
$\beth_i$	Termination trace for τ_i
$\delta_i^-(n)$	The minimum distance function of τ_i
$\eta_i^+(\Delta t)$	The maximum arrival function of τ_i
U_i	Utilization of τ_i
U	Utilization of a resource
$\mathcal{Z}$	Task set
$\mathcal{T}$	Typical task set
$\mathcal{O}$	Overload task set
$\mathcal{I}_i$	Interfering task set with τ_i
$dmm_i(k)$	DMM of τ_i

Table 2.1: Table of notations.

occupy the resource. After being scheduled, τ_i executes until completion. One execution of a task is called *instance* or *job*. During the execution, τ_i is subject to be blocked or preempted by other tasks according to the scheduling policy that applies to the resource. When multiple instances of τ_i are pending (ready to execute), instances are stored in a *ready queue* and executed under the First-In-First-Out (FIFO) manner. We call the tasks that can preempt or block the execution of τ_i as *interfering tasks*. To describe the patterns of incoming event streams that activate a task, the notion of traces will be presented.

Definition 2.5 (Activation trace)**.** *An activation trace $\aleph_i$ of a task τ_i is a function*

$$\aleph_i : \mathbb{N}^+ \to \mathbb{R}^+ \tag{2.1}$$

where $\aleph_i(n) = t$ indicates the absolute time at which the n-th instance of τ_i is activated.

Definition 2.6 (Termination trace)**.** *A terminations trace $\beth_i$ of a task τ_i is a function*

$$\beth_i : \mathbb{N}^+ \to \mathbb{R}^+ \tag{2.2}$$

where $\beth_i(n) = t$ indicates the absolute time at which the n-th instance of τ_i is terminated (completes execution).

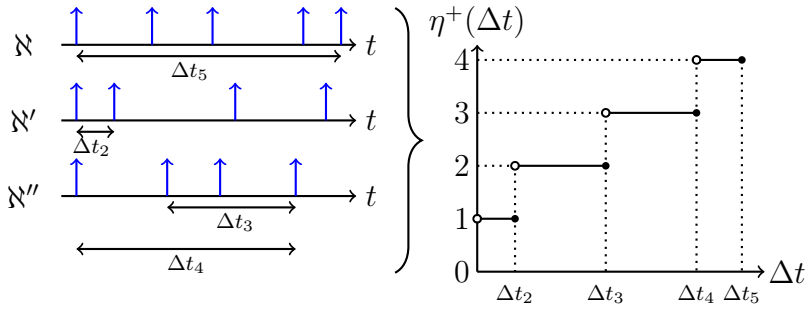

Figure 2.2: Relation of activation traces and the maximum arrival function η^+, based on [82].

The activation model is an abstraction of the activation trace. In this thesis we describe activation models using arrival curves.

Definition 2.7 (Arrival curves)**.** *The maximum and minimum arrival curves $\eta_i^+(\Delta t)$ and $\eta_i^-(\Delta t)$ of task τ_i are functions $\mathbb{R}^+ \to \mathbb{N}^+$ such that for any half-open time interval $[t, t+\Delta t)$, $\eta^+(\Delta t)$ ($\eta^-(\Delta t)$) defines the maximum (minimum) number of events occur during this time interval.*

Figure 2.2 shows how to abstract the maximum arrival function η^+ from activation traces. Arrival functions are non-decreasing [64]. In addition, the maximum arrival functions are sub-additive [64], i.e.,

$$\forall \Delta t, \Delta t' \in \mathbb{R}^+ : \eta^+(\Delta t) + \eta^+(\Delta t') \geqslant \eta^+(\Delta t + \Delta t') \tag{2.3}$$

The pseudo-inverse of arrival curves, distance functions, can be used as well to describe the activation models.

Definition 2.8 (Distance functions)**.** *The minimum and maximum distance functions $\delta_i^-(n), \delta_i^+(n) : \mathbb{N}^+ \to \mathbb{R}^+$ of task τ_i returns the minimum and maximum time intervals, respectively, during which at most n events occur.*

Arrival curves can be derived from distance functions as follows [32]:

$$\Delta t = 0 \quad : \quad \eta^+(\Delta t) = 0 \tag{2.4}$$

$$\Delta t > 0 \quad : \quad \eta^+(\Delta t) = \max_{n \geqslant 1, n \in \mathbb{N}} \{n | \delta^-(n) < \Delta t\} \tag{2.5}$$

$$\eta^-(\Delta t) = \min_{n \geqslant 0, n \in \mathbb{N}} \{n | \delta^+(n+2) < \Delta t\} \tag{2.6}$$

Figure 2.3 shows the minimum distance function $\delta^-(n)$ as a pseudo-inverse of

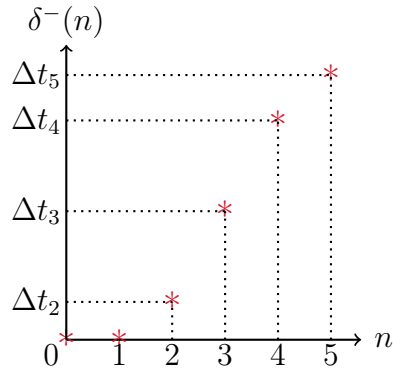

Figure 2.3: The minimum distance function $\delta^-(n)$ as a pseudo-inverse of the maximum arrival function $\eta^+(\Delta t)$ in Figure 2.2.

the maximum arrival function $\eta^+(\Delta t)$ in Figure 2.2. Minimum distance functions are non-decreasing and super-additive, i.e.,

$$\forall n, n' \in \mathbb{N}^+ : \delta^-(n) + \delta^-(n') \leqslant \delta^-(n + n') \tag{2.7}$$

Note that another version of arrival curves $\tilde{\eta}_i^+(\Delta t)$ and $\tilde{\eta}_i^-(\Delta t)$ has been defined in [99]. In $\tilde{\eta}_i^+(\Delta t)$ and $\tilde{\eta}_i^-(\Delta t)$ the considered time intervals are closed. While the analysis for Fixed Priority Preemptive (FPP), for instance, uses right-open intervals, the analysis for Earliest Deadline First (EDF) scheduling is based on closed intervals to avoid the phenomenon of "not looking far enough" in the busy-window analysis [33, 27]. The busy-window analysis will be presented in the next chapter.

Next, we formally define a pending instance.

Definition 2.9 (Pending instance)**.** *For a given task τ_i, an instance n is said to be pending at time t if*

$$\aleph_i(n) \leqslant t < \beth_i(n).$$

Now we can define when the resource is *idle*, which is needed to identify the timing behavior of schedulers.

Definition 2.10 (Idle)**.** *A resource r is said to be idle at time t if there is no pending instance at t of all tasks executing on r.*

Definition 2.11 (Relative deadline)**.** *The relative deadline D_i of a task τ_i is the amount of time allowed after activating the task during which the task must complete.*

Definition 2.12 (Absolute deadline)**.** *The absolute deadline d_i^n is a point of time at which the n-th instance of a task τ_i has to complete execution.*

Definition 2.13 (Sporadic task)**.** *A task that is activated by events that rarely occur and at unpredictable time such that* $\delta^+(2) = +\infty$ *is called sporadic task.*

An example of such tasks is routines for handling externally generated exception.

Another interesting timing property for timing verification is the response time.

Definition 2.14 (Response time)**.** *The response time* R_i^n *of the* n*-th instance of task* τ_i *is the time elapsed between the activation and the completion of the instance*

$$R_i^n = \beth_i(n) - \aleph_i(n) \tag{2.8}$$

Definition 2.15 (Worst-case response time)**.** *The worst-case response time* R_i^+ *of task* τ_i *is the longest response time that* τ_i *may experience*

$$\forall n \in \mathbb{N}^+ : R_i^+ \geqslant R_i^n \tag{2.9}$$

Chapter 3 illustrates how to compute the worst-case response time using the arrival curves.

When a scheduling policy, which defines how the contention between tasks will be solved, can schedule a given task set $\mathcal{Z}$ such that all tasks meet their deadline ($R_i^+ \leqslant D_i$), we say the scheduling policy can find a *feasible schedule.*

Definition 2.16 (Interfering task set)**.** *The set of interfering tasks* $\mathcal{I}_i \subseteq \mathcal{Z}$ *is a set of tasks that can preempt or block the execution of* τ_i*.*

2.1.2 Weakly-hard real-time system

Weakly-hard real-time systems and missing deadlines are intrinsically linked, and missing deadlines are also intrinsically linked with overload conditions. Therefore, we start with defining the overloaded systems.

Definition 2.17 (deadline-miss)**.** *A task* τ_i *has a deadline-miss if and only if*

$$\exists n \in \mathbb{N}^+ : R_i^n > D_i \tag{2.10}$$

Definition 2.18 (Utilization)**.** *The computation of utilization of a task* τ_i *requires the evaluation a limit for time approaching infinity.*

$$U_i = \lim_{\Delta t \to \infty} \frac{\eta_i^+(\Delta t) \ . \ C_i}{\Delta t} \tag{2.11}$$

the resource utilization is then:

$$U = \sum_{i \in \mathcal{Z}} U_i \tag{2.12}$$

Note that the definition of utilization is equivalent to the definition of maximum load in [32].

Definition 2.19 (Temporarily overloaded resource)**.** *A resource is temporarily overloaded if the resource utilization $U \leqslant 1$ and there is at least one deadline-miss takes place.*

Activating sporadic tasks such as exception handling routines may perturb the execution of other tasks, leading to deadline-misses. The sporadic tasks represent in this case a *transient overload*, which causes a temporarily overloaded resource.

Definition 2.20 (Temporarily overloaded system)**.** *A real-time system that has at least one temporarily overloaded resource is a temporarily overloaded system.*

Following the hard real-time model, a temporarily overloaded system is not schedulable. However, missing a few deadlines in a sequence of instances of a task is admissible for many systems due to, e.g., functionality robustness or human perspective as has been shown in Section 1.2.2. Such tasks have an extra timing requirement beside the deadline to specify the admissible number of deadline-misses in a consecutive sequence of instances.

Definition 2.21 ((m, k) Constraint)**.** *An (m, k) constraint specifies the number of deadline-misses $m \geqslant 0$ that is admissible for a task τ_i in any k consecutive instances where $k \geqslant m$.*

Let us now discuss how to specify (m, k) constraints based on safety. When a transient overload causes a deadline-miss, it is said to be an *active fault* and the deadline-miss is an *error*. The error may cause a *failure*, i.e., the system delivers an incorrect service [4]. If the error causes no failure, it is said to be *tolerable*. For a wide range of real-time systems not every individual deadline-miss will cause a failure and a task can tolerate therefore a few deadline-misses with no need for error handling.

To illustrate this point let us focus on control engineering. The deadline of a controller task is defined as the associated delay of the minimized cost function that guarantees the intended performance [73] [23]. This deadline is called *performance deadline* [73]. Missing one performance deadline means that the controller output will not be updated, however, not refreshing the output for a period may cause a failure related to a deviation from the intended controlled system space trajectory.

Hence, we say the controller task can tolerate few deadline-misses (say m) as long as it refreshes the plant input before another deadline-miss takes place and causes a failure. We define next the phrase "τ_i tolerates deadline-misses".

Definition 2.22. *A task τ_i tolerates— without error handling— at most m deadline-misses out of any k consecutive instances and no failure will be triggered unless τ_i misses $m+1$ deadlines out of any k consecutive instances where $m \geqslant 0$ and $k \geqslant m$.*

When a specific level of Quality-of-Service (QoS) or other metrics (cost, energy consumption, etc.) is required then another proper (m, k) constraint can be defined.

The definition of hard real-time systems can be derived by saying it is a system in which $m = 0$ for all tasks. A real-time system whose tasks can tolerate a few deadline-misses in a sequence of instances is known as Weakly-Hard Real-Time (WHRT) system.

Definition 2.23 (WHRT task)**.** *A WHRT task is a task that tolerates a precisely bounded number of deadline-misses in a sequence of k instances*[1]*.*

Definition 2.24 (WHRT system)**.** *A WHRT system is a system that comprises at least one WHRT task.*

Note that this definition is tighter than the original definition in [7] because it restricts the class of weakly-hard to tasks that can tolerate deadline-misses. In the rest of this thesis, *WHRT task* and *WHRT system* refer to the new definitions: Definition 2.23 and Definition 2.24, respectively.

This thesis considers a WHRT single-resource (uniprocessor) system in which the transient overload is due to sporadic tasks, e.g., interrupt service routine.

The transient overload that is due to extra sporadic instances, like cyclic-and-spontaneous messages in a CAN bus, is not considered in this thesis. Considering this kind of transient overload requires an analysis to extract the overload instances from the activation and termination traces. This problem is addressed in [2, 60].

2.1.3 Weakly-hard schedulability

Verifying timing requirements of a given task set is the purpose of performing a schedulability analysis. Classically, a system $\mathbb{S}$ is said to be schedulable if all its tasks meet their deadline. Suppose that a task set $\mathcal{Z}$ is given, the problem is then

[1] For the sake of brevity, we call such a sequence of k instances a *k-sequence*.

to verify that $\forall \tau \in \mathcal{Z}$ there is no deadline-miss. For WHRT systems, this condition has to be updated to consider the (m, k) constraints.

Definition 2.25 (Weakly-Hard Schedulability)**.** *A WHRT task is said to be weakly-hard schedulable if it meets its (m, k) constraint.*

Worst-case response time analyses, which are used to verify the worst-case response time against the deadline, cannot be used to verify the weakly-hard schedulability. Therefore, we need an analysis to compute upper bounds on the number of deadline-misses that a task may miss in a k-sequence and verify them against the (m, k) constraints.

WHRT systems need two essential analyses: 1) specifying the (m, k) constraints 2) verifying the (m, k) constraints. **This work focuses on verifying the (m, k) constraints and it does not address the problem of specifying these constraints**. Specifying the constraints needs an application specific analysis to study the impact of deadline-misses on system functions. Recent work [39, 12, 85] in this direction indicate that this question is indeed considered as relevant in the research community as well as in the industry.

2.2 Problem Statement

In this thesis, we consider that the task set $\mathcal{Z}$ consists of two subsets: 1) the sporadic overload tasks that represent the transient overload, we call this subset the *overload task set* $\mathcal{O}$; 2) the other tasks, we refer to them as the *typical task set* $\mathcal{T}$. That is to say, $\mathcal{Z} = \mathcal{T} \cup \mathcal{O}$. We call a task $\tau_j \in \mathcal{O}$ an *overload task*, and its instances *overload instances*. Intuitively, within the time intervals where none of the overload tasks are activated the system is not overloaded and no deadline-misses occur.

The classification of tasks into typical and overload is possible in the considered system model because the overload is due to internal overload conditions, i.e., interrupt service routine, exception handling routines, recovery tasks and reconfiguration tasks. However, such a straightforward classification is not possible for all temporarily overloaded system. In cases where such a distinction is not possible, an algorithm to identify the overload tasks or overload instances is required [60].

For a given task $\tau_i \in \mathcal{T}$, the main contribution of the thesis addresses the problem of how to upper bound the number of deadline-misses that may occur within a k-sequence of τ_i. For that end, we show how to compute a *Deadline Miss Model (DMM)* for a given task.

Definition 2.26. *A deadline miss model DMM consists of a function*

$$DMM \;=\; < dmm_i(k) > \tag{2.13}$$

with

$$dmm_i : \mathbb{N}^+ \rightarrow \mathbb{N} \tag{2.14}$$

which returns an upper bound on the number of deadline-misses out of any k-sequence.

We assume, for simplicity, that deadline miss models and their related functions are indexed according to their associated task. In this context, dmm_i is the deadline miss model for task τ_i.

In this thesis, a *deadline-miss-agnostic* scheduling is assumed. Such a scheduler lets all tasks run to completion even if a deadline-miss occurs. Also, an infinite ready queue is assumed, thereby, there will be no overload reduction. Studying the impact of killing the running task instance due to a deadline-miss or dropping the next task instance/instances due to an enforcement policy defined in the scheduling algorithm or a finite ready queue is out of the scope of this thesis. The reader, who is interested in the case of finite ready queues, may check [3].

2.3 Summary

This chapter elaborated formally what a temporarily overloaded system is, and it offered clean and precise definitions about WHRT systems, constraints, schedulability analysis and guarantees. Later, the addressed problem in this thesis is formally presented.

Considering the presented system model, Chapter 3 shows how to compute DMMs for tasks under different scheduling policies. Each considered scheduling policy will be presented thoroughly in Chapter 3. Therefore, this chapter introduced only the principal concept behind scheduling policies.

Chapter 5 will add some specifications to the presented system model. These specifications, namely task dependencies, will be elaborated formally.

3 COMPUTATION OF DMMs FOR INDEPENDENT WHRT TASKS

In this chapter, we present how to compute Deadline Miss Models (DMMs) for independent WHRT tasks scheduled on one resource considering different scheduling policies. The DMM is computed for a given task τ_i and it upper bounds the number of deadline-misses that may occur within a sequence of k consecutive instances of τ_i. We based our work on Typical Worst-Case Analysis (TWCA) [91].

TWCA computes a less-than-worst-case response time for a given task τ_i. It also computes the maximum number of deviations from this new bound. Therefore, there is a trivial DMM based on TWCA when it is applied to temporarily overloaded systems. In this chapter, we first describe TWCA in a nutshell. Next, we show how we improved over state-of-the-art TWCA to compute DMMs for FPP, FPNP, WRR, and EDF scheduling policies.

We elaborate on how to compute DMMs for FPP and FPNP scheduling policies in Section 3.2. In that section, we show that the problem of computing DMMs is a multidimensional knapsack problem. Sections 3.3 and 3.4 show how to adopt the proposed analysis to compute DMMs for WRR and EDF scheduling policies respectively. The related work is addressed thoroughly in Section 3.5.

This chapter presents work that has been published in [50], [125], [48] as well as [51] that is submitted to ACM TECS.

Notation 3.1. *The analysis that is presented in this chapter is applicable for computing and communication resources. When a communication resource is considered, it is proper to use the term "message" instead of "task" and "transmission time" instead of "execution time". However, the terms task and execution time are used for generality.*

3.1 Typical Worst-Case Analysis in A Nutshell

Methods addressing the worst-case timing scenario for a given real-time system have been presented in Chapter 1. Results provided by these methods do not reflect the frequency of worst-case occurrences even though, practically, the system may rarely experience the worst-case scenario.

TWCA [91, 92] was introduced to bound the frequency of worst-case scenario occurrences. The idea is to identify system behaviors considered as *typical-case* and to consider remaining scenarios as the result of some sporadic overload. That is to say the *non-typical* timing scenario is due to sporadic tasks, e.g., interrupt service routines or due to extra sporadic instances like cyclic-and-spontaneous messages in a CAN bus without causing any deadline-miss. Figure 3.1 shows sporadic instances

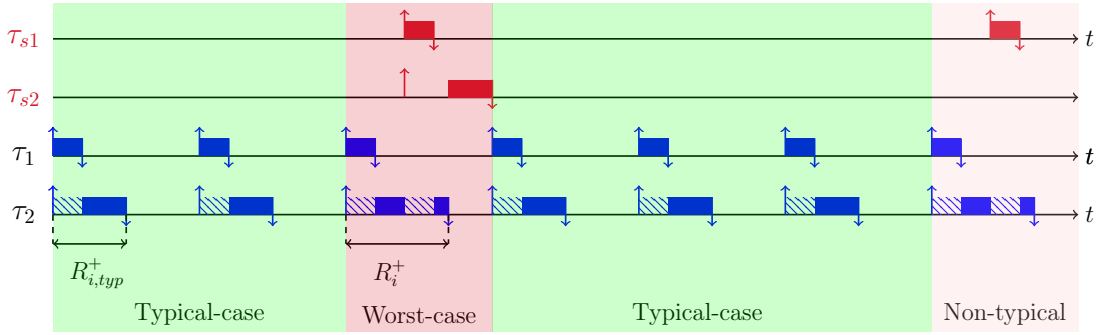

Figure 3.1: TWCA identifies system behaviors considered as typical-case and it considers remaining scenarios (non-typical) as the result of some sporadic overload. The red upward arrows indicate a sporadic overload instants.

in red. The absence of sporadic instances identifies the typical-case scenario, while the presence of them identifies the non-typical scenario.

Besides the worst-case response time, TWCA computes for a given task τ_i the longest response time considering the typical-case scenario, i.e., with absence of sporadic instances. This bound is called typical worst-case response time and denoted $R^+_{i,typ}$[1]. Also, TWCA computes the maximum number of deviations from $R^+_{i,typ}$ which τ_i may experience in a time window of k consecutive executions. This bound is called error model and denoted $err(k)$. The error model is an upper bound on the frequency of worst-case occurrences. In the example of Figure 3.1, task τ_2 experiences the worst-case scenario only once with $R^+_i = 3.5$. In most cases, the response time of τ_2 is bounded with $R^+_{i,typ} = 2.5$ and within a window of $k = 7$ executions, the response time of τ_2 diverges from $R^+_{i,typ}$ only twice ($err_i(7) = 2$).

[1]It is denoted *TWCRT* in the original work [91].

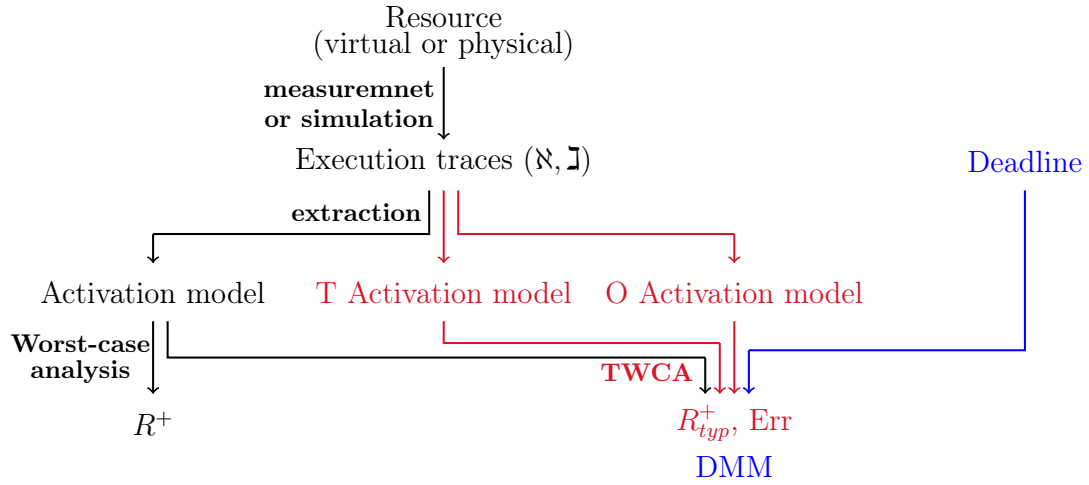

Figure 3.2: TWCA in Design Process.

Figure 3.2 illustrates the inputs and outputs of TWCA. TWCA uses execution traces (activation and termination traces). For a given task τ_i, TWCA computes $(R^+_{i,typ}, err_i(k))$. The computation of $R^+_{i,typ}$ and $err_i(k)$ is not shown in this thesis.

TWCA does not consider the concept of the deadline, instead, its main concern is to overcome over-provisioning for systems, in which the worst-case scenario is the result of some rarely activated sporadic instances. However, there is a trivial DMM based on TWCA when it is applied to temporarily overloaded systems. Computing $err_i(k)$ as the maximum number of deviations from D_i instead of $R^+_{i,typ}$, makes $err_i(k)$ as a DMM for τ_i. This work is extending TWCA using response time dependencies to compute tighter DMMs than $err_i(k)$.

3.2 DMM for Fixed Priority Scheduling Policies

In this section, we present our analysis to compute DMMs for FPP and FPNP. TWCA is a response time based analysis, thus, the worst-case response time analysis represents a foundation for it. The concept of *busy-window* is a cornerstone in computing a DMM using TWCA. For that reason we present first the worst-case response time analysis, then we show how to compute a DMM for a given task τ_i. The DMM computation is formulated as an integer linear programming (ILP) problem and it will be relaxed to a linear programming (LP) problem for better efficiency.

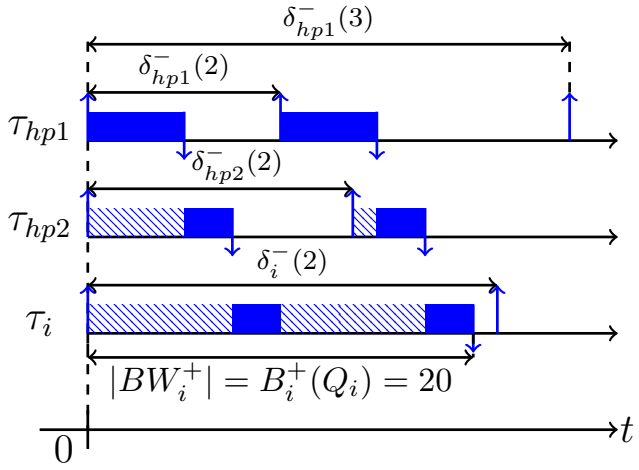

Figure 3.3: The longest τ_i busy-window under FPP scheduling policy. The critical instant is at $t = 0$.

3.2.1 FPP and FPNP scheduling policies

Tasks in the task set $\mathcal{Z}$ are assigned *fixed priorities*, i.e., the assigned priority π_i to $\tau_i \in \mathcal{Z}$ will never change as the system is running. An instance of τ_i occupies the resource when it has the highest priority among all ready instances. In FPP when an instance of another task τ_j with a higher priorityn than τ_i becomes ready, the instance occupying the resource will be preempted in the favor of the higher priority ready instance and it resumes when it is the highest ready instance again. In FPNP an instance will not be preempted till completion.

3.2.2 Worst-case response time analysis

We depend on the *busy-window* technique [65, 118]. The purpose of this section is to show how to compute for a given task τ_i the length of the longest (worst-case) τ_i busy-window, denoted BW_i^+, and the worst-case response time R_i^+.

Definition 3.1. *For a given task τ_i, a τ_i busy-window BW_i[2] is a maximal time interval $[t_1, t_2[$ where $\forall t \in [t_1, t_2[$ the resource is busy executing instances of τ_i and interfering tasks $\mathcal{I}_i$.*

The longest busy-window (worst-case busy-window) might not be observable on execution traces (activation and termination traces). Therefore, we need to produce it synthetically to obtain the *critical instant* [69], i.e., the point of time at which τ_i and all interfering tasks are simultaneously activated. Lehoczky [65]

[2]It is called in some literatures as a level-i busy interval [118].

generalized the concept of critical instant, and introduced the concept of busy-window. Later, Thiele et al. [116] proved the correctness of producing the worst-case busy-window using arrival curves. We obtain the critical instant by aligning the arrival curves of τ_i and all interfering tasks such that the tasks are activated simultaneously. Figure 3.3 shows the critical instant of three tasks.

Let us start with FPP. The set of interfering tasks $\mathcal{I}_i$ contains tasks of priority higher or equal to τ_i, Figure 3.3 shows the longest τ_i busy-window. In literature, the length of the longest τ_i bus-window BW_i^+ is computed based on the function $B_i^+(q)$ that bounds the maximum time needed to process q consecutive instances of a given task τ_i.

Lemma 3.1. *For a given task $\tau_i \in \mathcal{Z}$ with FPP, $B_i^+(q)$ is computed as follows:*

$$B_i^+(q) = q \ . \ C_i + \sum_{j \in hp(i)} \eta_j^+(B_i^+(q)) \ . \ C_j \tag{3.1}$$

where $hp(i)$ is the set of tasks that have a priority higher or equal to τ_i:

$$hp(i) = \{j \mid \pi_j \geqslant \pi_i, \forall j \in \mathcal{Z}\} \tag{3.2}$$

Proof. The proof is presented in [107] and it depends on the argumentation in [118]. The q-th instance will have to wait until all $(q-1)$ instances are executed to completion requesting their worst-case execution time (first summand). The q-th instances will be scheduled when it has the highest priority among all ready instances, thus, it will have to wait until all ready instances of higher priority tasks execute to completion requesting their worst-case execution time (second summand). By Definition 2.7 $\eta_j^+(\Delta t)$ returns the maximum number of instances that may be activated in a duration. Therefore, q instances of τ_i require at most $B_i^+(q)$ to be fully processed. □

If $B_i^+(q) \leqslant \delta_i^-(q+1)$, then the busy-window that include q instances does not include the $q+1$-th instance. This can be used to determine the maximum number Q_i of instances of τ_i in a τ_i busy-window.

$$Q_i = \min\{q \geqslant 1 \mid B_i^+(q) \leqslant \delta_i^-(q+1)\}. \tag{3.3}$$

The length of the longest τ_i busy-window (it is called also the worst-case τ_i busy-window) can then be computed as

$$|BW_i^+| = B_i^+(Q_i) \tag{3.4}$$

The analysis of FPNP is more complex due to boundary effects [27]. Here we show the equations needed to compute the length of the longest busy-window

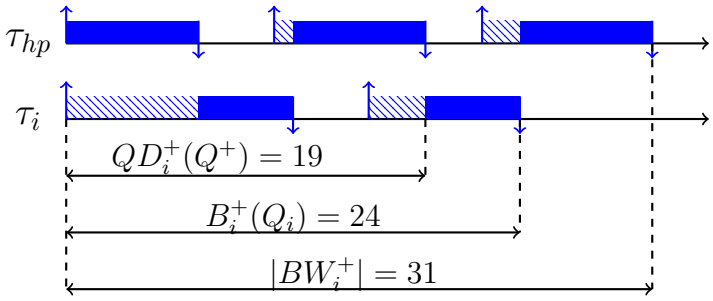

Figure 3.4: The longest τ_i busy-window under FPNP scheduling policy.

and the worst-case response time. The set of interfering tasks $\mathcal{I}_i$ contains tasks of priority higher or equal to τ_i and the task that has the longest worst-case execution time among tasks of lower priority than τ_i. We omit the proofs and further discussion as they are elaborated in [27, 5, 32].

For a given task τ_i, BW_i^+ is computed depending on $w_i(q)$:

$$w_i(q) = (q-1) \; . \; C_i + \max_{j \in lp(i)} \{C_j\} + \sum_{j \in hp(i)} \eta_j^+(w_i(q)) \; . \; C_j \tag{3.5}$$

where $\max_{j \in lp(i)}\{C_j\}$ is the maximum blocking time from lower priority tasks. $lp(i)$ is the set of tasks of lower priority than τ_i:

$$lp(i) = \{j \mid \pi_j < \pi_i, \forall j \in \mathcal{Z}\} \tag{3.6}$$

The maximum number Q_i of instances in one busy-window is:

$$Q_i = \min\{q \geqslant 1 \mid w_i(q+1) \leqslant \delta_i^-(q+1)\}. \tag{3.7}$$

Then the length of the longest busy-window is computed as follows:

$$|BW_i^+| = w_i(Q_i + 1) \tag{3.8}$$

The maximum processing time $B_i^+(q)$ of q consecutive instances is given as:

$$B_i^+(q) = QD_i(q) + C_i \tag{3.9}$$

where $QD_i^+(q)$ is the longest queuing delay that the q-th instance may experience.

$$QD_i^+(q) = (q-1) \; . \; C_i + \max_{j \in lp(i)} \{C_j\} + \sum_{j \in hp(i)} \tilde{\eta}_j^+(QD_i^+(q)) \; . \; C_j \tag{3.10}$$

$\tilde{\eta}^+$ needs to be considered due to boundary effects [27]. Note that using $\tilde{\eta}^+(QD_i^+(q))$ is equivalent to say $\eta^+(QD_i^+(q) + \varepsilon)$ as mentioned in [5] [32]. Figure 3.4 shows the longest τ_i busy-window and QD_i^+.

For both FPP and FPNP, the response time of every instance of τ_i is computed by

$$R_i^q = B_i^+(q) - \delta_i^-(q) \tag{3.11}$$

Theorem 3.1. *The worst-case response time of τ_i is computed by*

$$R_i^+ = \max_{1 \leqslant q \leqslant Q_i} \{B_i^+(q) - \delta_i^-(q)\} \tag{3.12}$$

Proof. It is proved in [32]. Any q instances require at most $B_i^+(q)$ to finish, and the q-th instance arrives no earlier than $\delta_i^-(q)$ after the first instance in BW_i^+. Thus, the difference is an upper bound on the response time of the q-th instance of τ_i in all τ_i busy-windows. Hence, the maximum of response times of instances in BW_i^+ is the worst-case response time. □

3.2.3 Deadline miss model computation

In this section we show how to compute a DMM for given task τ_i. The idea is to exploit the concept of busy-window, which bounds the impact of transient overload (sporadic tasks) in one busy-window.

Property 3.2. *In a given execution trace, a τ_i busy-window satisfies:*

$$\forall j \in \mathcal{I}_i : \aleph_j(q') \in BW_i \rightarrow \beth_j(q') \in BW_i$$

When an instance of the interfering task τ_j is activated within BW_i, then BW_i closes after this instance terminates. Remember that $\aleph_j$ indicates an activation trace of task τ_j and $\beth_j$ indicates a termination trace.

The above property implies that when sporadic instances (the transient overload) are activated in a busy-window BW_i, their impact, i.e., missing deadlines, is bounded in BW_i. Therefore, this property of the busy-window will be exploited to bound the DMM by:

1. bounding the number of deadline-misses of τ_i within one τ_i busy-window, let N_i denote this bound.

2. bounding the number of τ_i busy-windows in which instances of the k-sequence miss their deadline, let NBW_i^{miss} denote this bound.

Then, $dmm_i(k)$ can be bounded safely as

$$dmm_i(k) := N_i \ \times \ NBW_i^{miss} \tag{3.13}$$

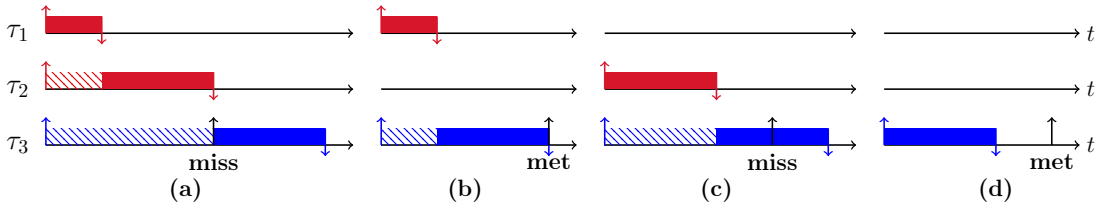

Figure 3.5: The impact of overload task combinations on τ_3 busy-windows: $\bar{c}_1 = \{\tau_1, \tau_2\}$ in (a), $\bar{c}_2 = \{\tau_1\}$ in (b), $\bar{c}_3 = \{\tau_2\}$ in (c) and $\bar{c}_4 = \{\emptyset\}$ in (d). The black arrow indicates D_3 and the red color indicates the overload tasks (τ_1, τ_2). Note that $N_3 = 1$.

Lemma 3.2. *Let*

$$N_i := \#\{q \mid 1 \leqslant q \leqslant Q_i \wedge R_i^q > D_i\} \tag{3.14}$$

Then N_i is an upper bound on the number of deadlines that τ_i may miss within one τ_i busy-window.

Proof. The argumentation is twofold. Firstly, the longest τ_i busy-window contains the maximum number Q_i of instances within one τ_i busy-window.

Secondly, the longest τ_i busy-window maximizes the response time of all instances $1 \leqslant q \leqslant Q_i$. That is because BW_i^+ includes the largest number of instances of τ_i and higher priority tasks within a duration of $|BW_i^+|$. And all higher priority instances are considered to request their worst-case execution time.

As a conclusion, the longest τ_i busy-window contains the maximum number of deadline-misses that τ_i may experience within any τ_i busy-window. □

Bounding NBW_i^{miss} is more complex. τ_i misses no deadline within a τ_i busy-window during which no overload instances execute, see Figure 3.5 (d). On the other hand, a τ_i busy-window experiences a deadline-miss when one or more overload instances are activated within the τ_i busy-window. Figure 3.5 (a), (b) and (c) illustrate the three possible scenarios in which a τ_3 busy-window will be impacted because of experiencing sporadic instances. The one can observe that τ_3 misses its deadline when a τ_3 busy-window experiences instances from τ_1 and τ_2 or only from $\{\tau_2\}$. Hence, we first identify the *combinations* of sporadic tasks that may cause a deadline-miss.

Definition 3.2. *A combination $\bar{c}$ is a subset of the overload task set $\mathcal{O}$. Then a combination could be a* **schedulable** *combination w.r.t. τ_i (causing 0 deadline-misses) or* **unschedulable** *combination*[3].

[3]A schedulable combination can be considered as a dormant fault while an unschedulable one is an active fault causing an error (deadline-miss).

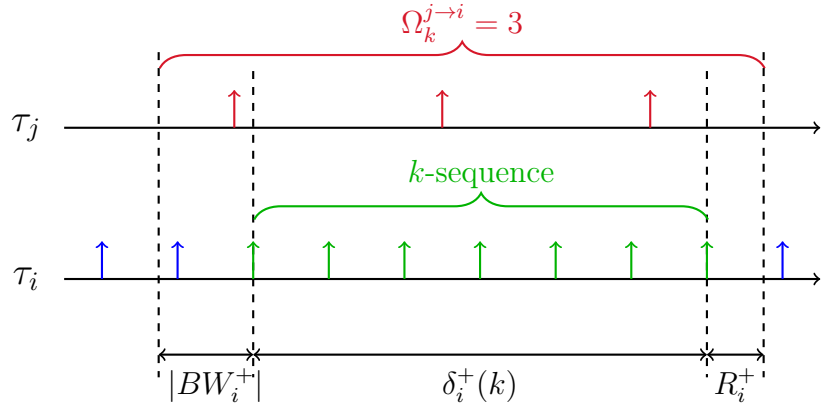

Figure 3.6: $\Omega_k^{j\to i}$ under FPP scheduling policy.

By $R_i^{\bar{c}}$ we denote the worst possible response time of τ_i in the task set $\mathcal{T} \cup \bar{c}$. In this way $R_i^+ = R_i^Z$. We say that a combination $\bar{c} \subseteq \mathcal{O}$ is unschedulable w.r.t τ_i if $R_i^{\bar{c}} > D_i$. In the same way we call the combination $\bar{c}$ for which $R_i^{\bar{c}} \leqslant D_i$ a schedulable combination. Let $\tilde{\mathcal{C}}_i$ denote the set of unschedulable combinations w.r.t. τ_i.

To bound NBW_i^{miss} we can compute an upper bound on the number of unschedulable combinations that may interfere with the considered k-sequence. For that end, we compute first an upper bound on the number of overload instances of τ_j that may interfere with any τ_i busy-window containing instances of the k-sequence.

Lemma 3.3. *$\Omega_k^{j\to i}$ is an upper bound on the number of overload instances of $\tau_j \in \mathcal{O}$ that may interfere with any τ_i busy-window containing instances of the k-sequence.*

$$\forall j \in \mathcal{O} \quad : \quad \Omega_k^{j\to i} := \eta_j^+(|BW_i^+| + \delta_i^+(k) + R_i^+) \qquad \textit{(FPP)} \tag{3.15}$$

$$\Omega_k^{j\to i} := \eta_j^+(|BW_i^+| + \delta_i^+(k) + (R_i^+ - C_i)) \quad \textit{(FPNP)} \tag{3.16}$$

Proof. Figure 3.6 illustrates $\Omega_k^{j\to i}$ where the k-sequence is shown in green and the sporadic instances are shown in red. The maximum arrival function $\eta_j^+(\Delta t)$ is an upper bound on the number of instances that may be activated within a duration Δt. The maximum arrival function is a non-decreasing function, thus, to compute $\Omega_k^{j\to i}$ as an upper bound we have to consider the maximum time window during which overload instances of τ_j may interfere with any τ_i busy-window containing instances of the k-sequence. $\delta_i^+(k)$ is the maximum distance between k instances.

BW_i^+: any overload instance shares the same τ_i busy-window with the first instance of the k-sequence, then it may impact its execution.

R_i^+: any overload instance that may be activated within the response time of

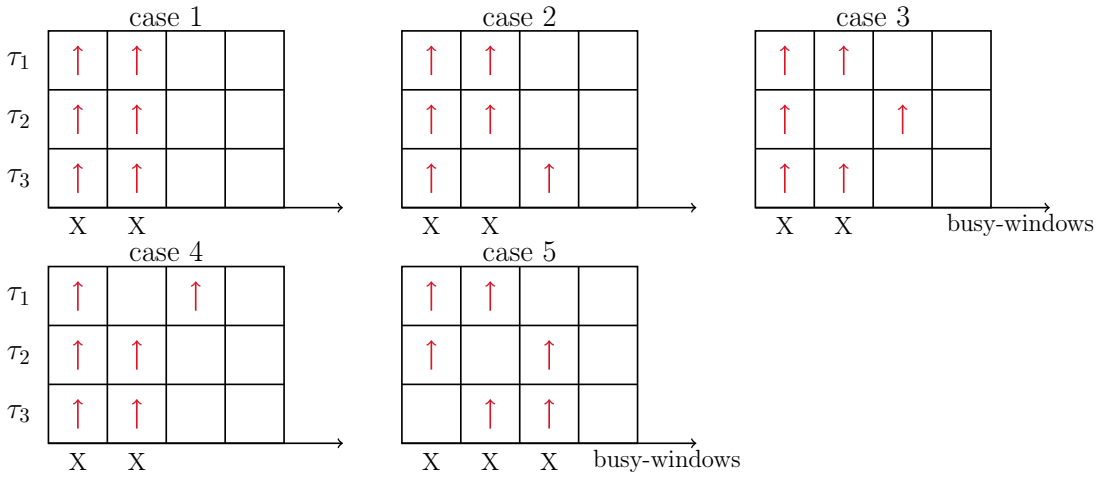

Figure 3.7: Packing overload instances into τ_4 busy-windows in Example 3.1. X indicates a deadline-miss.

the last instance of the k-sequence, may impact its execution (FPP). Because of the nonpreemptive policy in FPNP, it is sufficient to consider $R_i^+ - C_i$. □

For a given task τ_i and k, what we are looking for is to maximize the number of unschedulable combinations that can be composed using $\Omega_k^{j\to i}\ \forall j \in \mathcal{O} \cap hp(i)$.

Example 3.1. *Let us consider 4 tasks, $\pi_1 > \pi_2 > \pi_3 > \pi_4$, where τ_1, τ_2, τ_3 are overload tasks with $\Omega_k^{1\to 4} = 2$, $\Omega_k^{2\to 4} = 2$ and $\Omega_k^{3\to 4} = 2$. With 3 overload tasks we can combine 2^3 combinations, assume that the set of unschedulable combinations is $\tilde{\mathcal{C}}_4 = \{\bar{c}_1 = \{\tau_1, \tau_2, \tau_3\}, \bar{c}_2 = \{\tau_1, \tau_2\}, \bar{c}_3 = \{\tau_1, \tau_3\}, \bar{c}_4 = \{\tau_2, \tau_3\}\}$ and the other 4 combinations are schedulable. Using the given $\Omega_k^{j\to 4}$ values we can compose the unschedulable combinations in 5 different ways as Figure 3.7 illustrates. Case 5 has the maximum number of impacted busy-windows by maximizing the number of unschedulable combinations that can be composed using $\Omega_k^{j\to 4}$.*

Lemma 3.4. *Let $x_{\bar{c}}$ represent the number of composed instances of the unschedulable combination $\bar{c}$, then NBW_i^{miss} can be bounded as follows:*

$$NBW_i^{miss} \leqslant \max\left\{\sum_{\bar{c}\in\tilde{\mathcal{C}}_i} x_{\bar{c}}\right\} \tag{3.17}$$

with

$$\sum_{\bar{c}:j\in\bar{c}\in\tilde{\mathcal{C}}_i} x_{\bar{c}} \leqslant \Omega_k^{j\to i}\ \forall j \in \mathcal{O} \cap hp(i) \tag{3.18}$$

Proof. The overload instances within a window of k-sequence will be combined in unschedulable and schedulable combinations. Each unschedulable combination

$\bar{c}$ impacts no more than one τ_i bus-window, thus, with $x_{\bar{c}} \leqslant \min_{j \in \bar{c}}\{\Omega_k^{j \to i}\}$ instances of $\bar{c}$ there will be no more than $x_{\bar{c}}$ impacted busy-window. Upper bounding NBW_i^{miss} implies maximizing $\sum_{\bar{c} \in \tilde{\mathcal{C}}_i} x_{\bar{c}}$.

□

By substituting 3.17 and 3.18 in 3.13 we get the ILP formulation in 3.19 that can be used to compute a DMM for a given task $\tau_i \in \mathcal{T}$.

Theorem 3.3. *The ILP in 3.19 is a DMM for $\tau_i \in \mathcal{T}$ when FPP or FPNP scheduling policies are considered.*

$$dmm_i(k) := N_i \,.\, \max\Big\{ \sum_{\bar{c} \in \tilde{\mathcal{C}}_i} x_{\bar{c}} \;:\; \sum_{\bar{c}: j \in \bar{c} \in \tilde{\mathcal{C}}_i} x_{\bar{c}} \leqslant \Omega_k^{j \to i} \; \forall j \in \mathcal{O} \cap hp(i),\, x_{\bar{c}} \in \mathbb{N} \; \forall \bar{c} \in \tilde{\mathcal{C}}_i \Big\} \tag{3.19}$$

Proof. On the one hand, within one τ_i busy-window there will be no more than N_i deadline-misses as has been proven in Lemma 3.2.

On the other hand, there will be no more than $\max\Big\{ \sum_{\bar{c} \in \tilde{\mathcal{C}}_i} x_{\bar{c}} \Big\}$ impacted τ_i busy-windows (each includes at least one deadline-miss) as has been shown in Lemma 3.4.

The ILP in Equation 3.19 provides, therefore, an upper bound on the number of deadlines that τ_i might miss out of any k-sequence, i.e., a DMM. □

In Example 3.1 and Figure 3.7, there will be:

1. case 1: $x_{\bar{c}_1} = 2 \Rightarrow \sum_{\bar{c} \in \tilde{\mathcal{C}}} x_{\bar{c}} = 2$
2. case 2: $x_{\bar{c}_1} = 1, x_{\bar{c}_2} = 1 \Rightarrow \sum_{\bar{c} \in \tilde{\mathcal{C}}} x_{\bar{c}} = 2$
3. case 3: $x_{\bar{c}_1} = 1, x_{\bar{c}_3} = 1 \Rightarrow \sum_{\bar{c} \in \tilde{\mathcal{C}}} x_{\bar{c}} = 2$
4. case 4: $x_{\bar{c}_1} = 1, x_{\bar{c}_4} = 1 \Rightarrow \sum_{\bar{c} \in \tilde{\mathcal{C}}} x_{\bar{c}} = 2$
5. case 5: $x_{\bar{c}_2} = 1, x_{\bar{c}_3} = 1, x_{\bar{c}_4} = 1 \Rightarrow \sum_{\bar{c} \in \tilde{\mathcal{C}}} x_{\bar{c}} = 3$

Then $dmm_4(k) = 3 \times N_4$ is an upper bound on the number of deadline-misses that task τ_4 may miss out of any k consecutive instances.

The problem of computing a DMM as stated in the ILP 3.19 can be described as *packing* as many unschedulable combinations as possible into τ_i busy-windows, see Figure 3.7. Therefore, the problem becomes a *multidimensional knapsack problem.*

The multidimensional knapsack problem is defined by the following ILP [74]:

$$\text{maximize} \quad \sum_{j=1}^{N} p_j x_j \tag{3.20}$$

$$\text{subject to} \quad \sum_{j=1}^{N} a_{ij} x_j \leqslant b_i, \quad i = 1, \ldots, m \tag{3.21}$$

$$0 \leqslant x_j \leqslant u_j, \qquad j = 1, \ldots, N \tag{3.22}$$

$$x_j \in \mathbb{N}, \qquad j = 1, \ldots, N \tag{3.23}$$

In our ILP we have: all items $j = 1, \ldots, N$ have the same profit p_j, $a_{ij} = 1$, and $u_j = +\infty$.

The proposed ILP 3.19 is not an efficient solution for two reasons:

- Computing the set of unschedulable combinations $\tilde{\mathcal{C}}$ using the usual worst-case response time analysis is not efficient because it is a fixed-point problem (see Equation 3.10 and 3.1) and therefore it is of high complexity.
- Finding an approximate algorithm for the multidimensional knapsak problem is not only NP-Hard [74] but also might have an exponential size input because $\#\{\tilde{\mathcal{C}}\} \leqslant 2^{\#\{\mathcal{O}\}} - 1$[4].

In the next two subsections these limitations are tackled.

3.2.4 An efficient schedulability criterion

The proposed solution to compute $dmm_i(k)$ for a given task τ_i requires computing the set of unschedulable combinations $\tilde{\mathcal{C}}_i$. Using the worst-case response time analysis, which is presented in Section 3.2.2, to compute $\tilde{\mathcal{C}}_i$ is not efficient because it requires to compute the worst-case response time of τ_i considering a subset of tasks $\mathcal{Z}' = \mathcal{T} \cup \bar{c}$ for all possible combinations. Note that there are $2^{\#\{\mathcal{O}\}} - 1$ possible combinations and for each of which we will have to solve a fixed-point equation (Equation 3.1 for FPP and Equation 3.10 for FPNP). Therefore, we present next an efficient schedulability criterion to classify combinations into schedulable and unschedulable for both FPP and FPNP respectively.

As a first step, we apply the worst-case response time analysis to compute BW_i^+ and the response time R_i^q of all instances $\forall q \in [1, Q_i]$. Next, we define for q that has $R_i^q > D_i$ the following:

$$\rho_i^q(t) := \sum_{j \in hp(i)} \eta_j^+(t) \; . \; C_j + q \; . \; C_i - t \qquad \forall t \leqslant B_i^+(q) \tag{3.24}$$

[4]The combination $\bar{c} = \{\}$ is always schedulable.

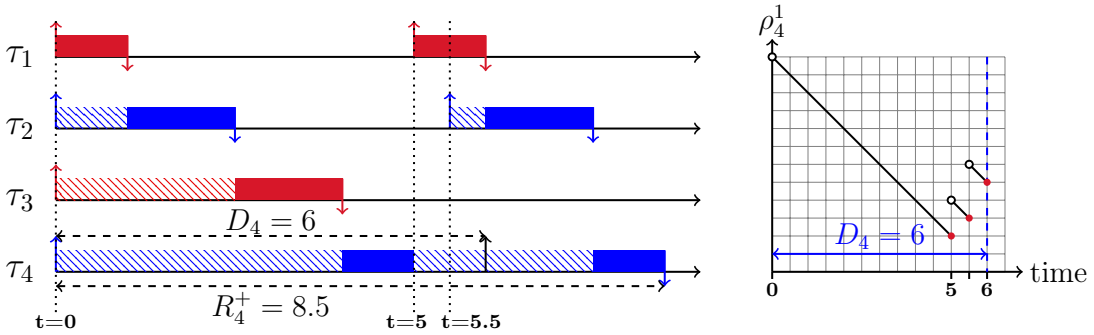

Figure 3.8: To make τ_4 finish before its deadline, we have to remove more workload than what remains to process at some time point.

$$wl_{j,over}(t) := \eta_j^+(t) \; . \; C_j \qquad \forall j \in \mathcal{O} \cap hp(i), \forall t \leqslant |BW_i^+| \tag{3.25}$$

$$\mathcal{P}_i^q := \{t \mid \delta_i^-(q) < t \leqslant \delta_i^-(q) + D_i\} \tag{3.26}$$

Intuitively, $\rho_i^q(t)$ refers to the workload in the worst-case τ_i busy-window that remains to be processed at time t before the q-th instance of τ_i finishes. In contrast, $wl_{j,over}(t)$ is the accumulated amount of overload from τ_j until time t. Note that these values do not consider the instances arriving at or after t. $\mathcal{P}_i^q$ corresponds to all the activation time points (from higher priority tasks) between the q-th instance of τ_i and its deadline, as well as the deadline point. In other words, $\mathcal{P}_i^q$ represents the time points where τ_i might finish its execution before the deadline. Figure 3.8 illustrates an example of 4 tasks with FPP. There are 4 points of time to check the schedulability of τ_4: $t = 0, t = 5, t = 5.5, t = 6$. $\rho_i(t)$ is a piecewise function. Within each piece, it decreases monotonically while $wl_{j,over}(t)$ remains the same. It is sufficient to analyze only the end point, indicated in red, of each piece, i.e., the local minima. In this example we have: $\rho_4^1(t = 5) = 1, \sum_{j\in\mathcal{O}} wl_{j,over}(t = 5) = 2.5, \rho_4^1(t = 5.5) = 1.5, \sum_{j\in\mathcal{O}} wl_{j,over}(t = 5.5) = 3.5, \rho_4^1(t = 6 = D_4) = 2.5, \sum_{j\in\mathcal{O}} wl_{j,over}(t = 6) = 3.5$.

The three values are defined similarly for FPNP:

$$\rho_i^q(t) := \sum_{j\in hp(i)} \eta_j^+(t) \; . \; C_j + (q-1) \; . \; C_i + \max_{j\in lp(i)}\{C_j\} - t \qquad \forall t \leqslant w_i(q) \tag{3.27}$$

$$wl_{j,over}(t) := \eta_j^+(t) \; . \; C_j \qquad \forall j \in \mathcal{O} \cap hp(i), \forall t \leqslant |BW_i^+| \tag{3.28}$$

$$\mathcal{P}_i^q := \{t \mid \delta_i^-(q) < t \leqslant \delta_i^-(q) + D_i - C_i\} \tag{3.29}$$

$\rho_i^q(t)$ refers to the workload in the worst-case τ_i busy-window that remains to be processed at time t before the q-th instance of τ_i starts. Here $\mathcal{P}_i^q$ represents the time points where τ_i might start its execution and finish before the deadline.

If a task $\tau_j : j \in \mathcal{O}$ and $j \notin \bar{c}$ then the overload due to τ_j is removed and therefore the workload that remains to be processed at time t before the q-th instance of τ_i finishes (FPP) respectively starts (FPNP) in the τ_i busy-window corresponding to $\bar{c}$ is $\rho_i^q(t) - wl_{j,over}(t)$ so for every instance q of τ_i, there must be a time point t before or equal to D_i (FPP) respectively $D_i - C_i$ (FPNP), so that $\rho_i^q(t) - \sum_{j \in \mathcal{O} \cap hp(i), j \notin \bar{c}} wl_{j,over}(t) \leqslant 0$.

A sufficient and necessary (FPP) respectively sufficient (FPNP) criterion to say that $\bar{c}$ is a **schedulable combination** is given then as follows:

$$\forall q \in [1, Q_i], R_i^q > D_i \; \exists t \in \mathcal{P}_i^q : \sum_{j \in hp(i), j \notin \bar{c}} wl_{j,over}(t) \geqslant \rho_i^q(t) \tag{3.30}$$

To justify why the above condition is only sufficient under FPNP, let $lp^{\bar{c}}(i)$ denote the set of lower priority tasks than τ_i in the task set $\mathcal{T} \cup \bar{c}$. That is:

$$lp^{\bar{c}}(i) = \{j \mid \pi_j < \pi_i, \forall j \in \mathcal{T} \cup \bar{c}\}$$

Intuitively, $lp^{\bar{c}}(i) \subseteq lp(i)$ because $\mathcal{T} \cup \bar{c} \subseteq \mathcal{Z}$. Therefore, $\max_{j \in lp(i)}\{C_j\} \geqslant \max_{j \in lp^{\bar{c}}(i)}\{C_j\}$, and $\rho_i^q(t)$ in Equation 3.27 is over-approximated. In this case, the condition in Equation 3.30 may not hold for a schedulable combination $\bar{c}$. However, when the condition in Equation 3.30 holds for a combination $\bar{c}$, then $\bar{c}$ is schedulable. That is to say, the condition in Equation 3.30 is sufficient under FPNP.

Consequently, the one can deduce that the set

$$\tilde{\mathcal{C}}_i := \{\bar{c} \mid \exists q \in [1, Q_i], R_i^q > D_i \; \forall t \in \mathcal{P}_i^q : \sum_{j \in hp(i), j \notin \bar{c}} wl_{j,over}(t) < \rho_i^q(t)\} \tag{3.31}$$

consists of all unschedulable combinations (FPP) respectively superset of unschedulable combinations (FPNP). In Figure 3.8, there is only one unschedulable combination $\bar{c}_3 = \{\tau_1, \tau_3\}$, thus, $\tilde{\mathcal{C}}_4 = \{\bar{c}_3\}$.

3.2.5 An efficient LP solution

As mentioned before, directly computing $dmm_i(k)$ is impractical. Instead, an LP relaxation is considered. $x_{\bar{c}}$ is an integer variable in 3.19, the LP relaxation considers this variable as continuous $x_{\bar{c}} \in \mathbb{R}^+$.

$$dmm'_i(k) = \max \; N_i \sum_{\bar{c} \in \tilde{\mathcal{C}}_i} x_{\bar{c}} \tag{3.32}$$

$$\text{s.t.} \sum_{\bar{c}:\, j \in \bar{c} \in \tilde{\mathcal{C}}_i} x_{\bar{c}} \leqslant \Omega_k^{j \to i} \;\; \forall j \in \mathcal{O} \cap hp(i) \tag{3.33}$$

$$x_{\bar{c}} \geqslant 0 \qquad\qquad \forall \bar{c} \in \tilde{\mathcal{C}}_i \;\; . \tag{3.34}$$

Trivially $dmm'_i(k) \geqslant dmm_i(k)$, so this defines a DMM as well.

N_i is a constant, thus, any unschedulable combination is considered to have the same impact. In other words, the above LP tries to use as much as possible of the minimal unschedulable combinations (assign the largest possible number to $x_{\bar{c}}$ when $\bar{c}$ is a minimal combination). In Example 3.1, $\bar{c}_2, \bar{c}_3, \bar{c}_4$ are minimal combinations while $\bar{c}_1$ is not; $dmm_i(k)$ is maximized only when $x_{\bar{c}_2}, x_{\bar{c}_3}, x_{\bar{c}_4}$ are maximized. Based on this observation we propose an algorithm using *column generation* that solves $dmm'_i(k)$, producing upper bounds in the process. The dual linear program of (3.32)–(3.34) reads as follows:

$$\min \sum_{j \in \mathcal{O} \cap hp(i)} \Omega_k^{j \to i} y_j \tag{3.35}$$

$$\text{s.t.} \sum_{j \in \bar{c},\, j \in \mathcal{O} \cap hp(i)} y_j \geqslant N_i \;\; \forall \bar{c} \in \tilde{\mathcal{C}}_i \tag{3.36}$$

$$y_j \geqslant 0 \qquad\qquad \forall j \in \mathcal{O} \;\; . \tag{3.37}$$

We start with a reduced set of unschedulable combinations to a smaller sample $W \subseteq \tilde{\mathcal{C}}_i$. Initially, this could be $W := \{\mathcal{O}\}$ ($\mathcal{O}$ is guaranteed to be unschedulable). Let x^* be an optimal primal solution, i.e., the values of $x_{\bar{c}}$ which report the maximum DMM, and y^* be an optimal dual solution, for this reduced LP, with objective value $z^* := N_i \sum_{\bar{c}} x^*_{\bar{c}}$.

In the generating columns based algorithm, we consider the LP with one variable (one column) and we aim to find a violated constraint of the complete dual LP to identify the next variable that will be added to the LP, which indicates the next unschedulable combination that will be added to W, by computing

$$\nu = \min_{\bar{c} \in \tilde{\mathcal{C}}_i} \sum_{j \in \bar{c},\, j \in \mathcal{O} \cap hp(i)} y^*_j \; . \tag{3.38}$$

When there is no more variables that violate the constraint of the complete dual LP, the column generation finishes. The next step is to identify whether z^* is a DMM (a single value for k):

Lemma 3.5. *If $\nu \geqslant N_i$, then x^* is optimal for 3.32–3.34, and $z^* \geqslant dmm_i(k)$.*

Proof. If the condition holds, then y^* is feasible for 3.35–(3.37). By LP duality, it follows that

$$z^* = dmm'_i(k) \geqslant dmm_i(k)$$

□

This lemma allows us to deduce a DMM when column generation finishes, i.e., a globally optimal solution has been found.

As this might take a long time due to the exponential size of $\tilde{\mathcal{C}}_i$, we seek to construct a DMM from suboptimal solutions as well. We define

$$dmm''_i(k) = \frac{N_i}{\nu} z^* \ . \tag{3.39}$$

Lemma 3.6. *If $0 < \nu \leqslant N_i$, then the obtained function dmm''_i is a DMM.*

Proof. From the construction of ν 3.38, it follows that

$$\sum_{j \in \bar{c}, j \in \mathcal{O} \cap hp(i)} \frac{N_i}{\nu} y^*_j \geqslant N_i$$

for every $\bar{c} \in \tilde{\mathcal{C}}_i$. Consequently, $\frac{N_i}{\nu} y^*$ is a feasible solution for 3.35–3.37, with objective value $\frac{N_i}{\nu} z^*$. Using weak LP duality, we get $dmm''_i \geqslant dmm'_i \geqslant dmm_i$, so dmm''_i is indeed a DMM. □

Algorithms. Before we can describe the overall algorithm, we discuss a subroutine to compute ν. Following the characterization of $\tilde{\mathcal{C}}_i$, we can rewrite 3.38, given y^*, as follows:

$$\nu = \min \sum_{j \in \mathcal{O} \cap hp(i)} y^*_j d_j \tag{3.40}$$

$$\text{s.t. } \exists q \in [1, Q_i], R^q_i > D_i \ \forall t \in \mathcal{P}^q_i : \sum_{j \in \mathcal{O} \cap hp(i)} wl_{j,over}(t)(1 - d_j) \leqslant \rho^q_i(t) - \varepsilon \tag{3.41}$$

$$d_j \in \{0, 1\} \tag{3.42}$$

using a sufficiently small $\varepsilon > 0$.

The overall algorithm to compute $dmm_i(k)$ is shown in Algorithm 1.

Algorithm 1: Get the approximate upper bound of LP (3.32)–(3.34)

$W = \{\mathcal{O}\}$, $dmm = k$
repeat
  Solve LP (3.32)–(3.34) (with W instead of $\tilde{\mathcal{C}}_i$), obtain primal solution x^*, dual solution y^*, objective value z^*
  Compute ν according to (3.40), let $\bar{c}$ be the combination attaining the minimum
  if $\nu \geqslant N_i$ **then**
    $dmm = z^*$
  else
    $W = W \cup \{\bar{c}\}$
    if $\nu > 0$ and $\frac{N_i}{\nu} z^* < dmm$ **then** $dmm = \frac{N_i}{\nu} z^*$
  output upper bound dmm
until $z^* = dmm$

3.3 DMM for WRR Scheduling Policy

In this section, we show how to compute DMMs for systems scheduled with a variant of Round-Robin (RR) called Weighted Round-Robin (WRR). This section is structured in the same way as previous one. A representation of WRR scheduling is given. Then, we recall the state-of-the-art worst-case response time analysis[99]. Finally, the DMM computation comes to light.

3.3.1 WRR scheduling policy

In operating systems, RR is considered as a preemptive extension of FIFO[5] [109]. However, it also can be seen as a work-conservative extension of Time-Division-Multiple-Access (TDMA) [36, 99]. RR scheduler enables a task $\tau_i \in \mathcal{Z}$ to occupy the resource for a predefined amount of time, called *time slot*[6] θ, before assigning the resource to the next tasks in a cyclic fashion. In this thesis, we are interested in WRR, which is a variant of RR. In WRR, each task has a different time slot θ_i based on intended share of the scheduled resource. The maximum resource cycle time, denoted WRR_{turn}, can be determined:

$$WRR_{turn} = \sum_{i \in \mathcal{Z}} \theta_i \tag{3.43}$$

[5]It is also called First-Come-First-Serve (FCFS) [109]
[6]It is also called time quantum or time slice [109]

When it is τ_i turn and there is no pending instance of τ_i, the resource will not be assigned to τ_i and the scheduling algorithm tests τ_{i+1}. Otherwise, τ_i occupies the resource with no preemption for at most θ_i units of time. If τ_i did not complete executing at the end of its time slot, it will be preempted by the scheduling algorithm, otherwise, τ_i is allowed to execute the next pending instances under the FIFO manner if there is any.

Whereas the classical round-robin is used for *soft* real-time systems or for *best efforts* QoS, WRR is used for hard real-time systems such as Asynchronous Transfer Mode (ATM) local area network [94] and Ethernet switches [115].

3.3.2 Worst-case response time analysis

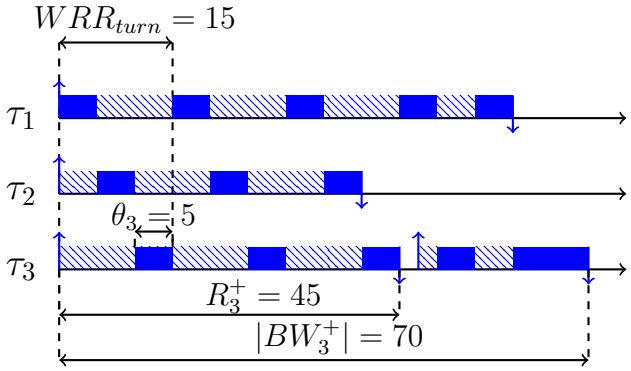

Figure 3.9: The longest τ_3 busy-window under WRR.

We aim to compute the length of the longest τ_i busy-window and the worst-case response time R_i^+ for a given task τ_i.

The critical instant, i.e., the starting point of the longest busy-window BW_i^+, is the point of time at which all tasks are simultaneously activated right after the time slot of τ_i has expired. The set of interfering tasks $\mathcal{I}_i = \mathcal{Z} \setminus \{i\}$ because WRR is a dynamic priority scheduling policy. We compute $|BW_i^+|$ iteratively starting with $|BW_i^+| = C_i$ as follows:

$$|BW_i^+| := \sum_{j \in \mathcal{Z}} \eta_j^+(|BW_i^+|) \cdot C_j \tag{3.44}$$

Lemma 3.7. *Equation 3.44 converges when the utilization* $U \leqslant 1$.

Proof. By Definition 2.17, any load will be fully processed within a bounded interval when $U \leqslant 1$. □

Let Q_i denotes the number of τ_i instances within BW_i^+, then

$$Q_i = \eta_i^+(|BW_i^+|) \tag{3.45}$$

To compute the worst-case response time R_i^+, we use the function $B_i^+(q)$ – as we did for FPP – that bounds the maximum time needed to process q consecutive instances of a given task τ_i. Figure 3.9 shows the longest τ_i busy-window.

Lemma 3.8. *For a given task τ_i with WRR, $B_i^+(q)$ is computed as follows:*

$$B_i^+(q) = q \;.\; C_i + \sum_{j \in \mathcal{Z} \setminus \{i\}} \min\left(\left\lceil \frac{q \;.\; C_i}{\theta_i} \right\rceil \times \theta_j \;,\; \eta_j^+(B_i^+(q)) \;.\; C_j \right) \tag{3.46}$$

Proof. The proof of this theorem follows the same argumentation in Lemma 3.1. The q-th instance will have to wait until all $(q-1)$ instances are executed to completion requesting their worst-case execution time (first summand). Each instance of τ_i needs to be scheduled at most $\lceil \frac{C_i}{\theta_i} \rceil$ times to complete execution. Therefore, an instance of τ_i gets interference from τ_j by at most $\lceil \frac{C_i}{\theta_i} \rceil \;.\; \theta_j$. However, if τ_j needs only $n \leqslant \lceil \frac{C_i}{\theta_i} \rceil$ time slots to complete execution and $C_j \leqslant n \;.\; \theta_j$, then an instance of τ_i gets interference from τ_j by C_j because WRR is a work-conserving scheduler. Processing q consecutive instances of τ_i gets interference from τ_j by $\min\left(\left\lceil \frac{q \;.\; C_i}{\theta_i} \right\rceil \times \theta_j \;,\; \eta_j^+(B_i^+(q)) \times C_j \right)$ (second summand). □

Theorem 3.4. *The worst-case response-time R_i^+ is then computed by*

$$R_i^+ = \max_{1 \leqslant q \leqslant Q_i} \{B_i^+(q) - \delta_i^-(q)\} \tag{3.47}$$

Proof. It is proved in Theorem 3.1. Because $B_i^+(q)$ is the maximum time to process q instances, and $\delta_i^-(q)$ is the minimum distance between these q, the response time of the q-th instance in BW_i^+ is an upper bound on the response time of the q-th instance in all τ_i busy-windows. Therefore, the worst-case response time of τ_i is the maximum of response times of instances in BW_i^+. □

3.3.3 Deadline miss model computation

In this section we show how to adopt TWCA to compute DMMs for real-time tasks sharing a resource under WRR scheduling.

Similar to Section 3.2, to compute a DMM:

- we compute an upper bound N_i on the number of deadlines that τ_i may miss within one busy-window;

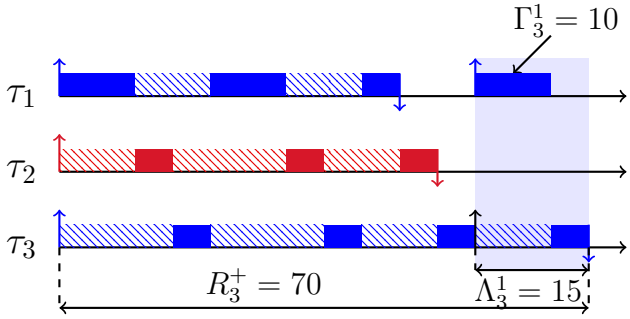

Figure 3.10: Λ_i and Γ_i where τ_2 is an overload task. The black upward arrow indicates D_3.

- we propose a criterion to compute efficiently the unschedulable combinations , see Definition 3.2;
- we compute an upper bound $\Omega_k^{j \to i}$ on the number of overload instances of τ_j that may interfere with any τ_i busy-window containing instances of the k-sequence.

Compute N_i.

Lemma 3.9. *Let*

$$N_i := \#\{q \mid 1 \leqslant q \leqslant Q_i \wedge R_i^q > D_i\} \tag{3.48}$$

Then N_i is an upper bound on the number of deadlines that τ_i may miss within one busy-window.

Proof. The proof of this theorem follows the same argumentation in Lemma 3.2. Response times of instances in BW_i^+ upper bound the response times of instances in all τ_i busy-windows. Therefore, the longest τ_i busy-window contains the maximum number of deadline-misses that τ_i may experience within any τ_i busy-window. □

Schedulability criterion. A task τ_i misses its deadline when it experiences interfering from certain combinations of overload tasks, those are known as unschedulable combinations. Other combinations, which do not drive τ_i to miss its deadline, are called schedulable combinations, see Definition 3.2. To compute DMMs using TWCA, we compute the set of unschedulable combinations. Just like FPP and FPNP, it is not efficient to use the worst-case response time analysis because it requires to solve a fixed-point equation (Equation 3.46) for each possible combination and therefore it is of high complexity.

Here, we present a sufficient condition on a combination $\bar{c}$ that guarantees the schedulability of the q-th instance of τ_i. Since we proved in Theorem 3.4 that

the worst-case response time is found in the longest τ_i busy-window BW_i^+, it is sufficient and necessary to apply our sufficient condition on instances in BW_i^+. Our approach starts with computing BW_i^+ and the response times of instances $\forall q \in [1, Q_i]$. When q misses its deadline, then it has a *lateness* Λ_i^l of:

$$\Lambda_i^q := R_i^q - D_i \tag{3.49}$$

Figure 3.10 shows the lateness of task τ_3. Let $wl_{j,over}^q$ denote the contribution of the overload task τ_j in $\delta_i^-(q) + D_i$:

$$wl_{j,over}^q := \min\left(\left\lceil \frac{q \,.\, C_i}{\theta_i} \right\rceil \times \theta_j \;,\; \eta_j^+(\delta_i^-(q) + D_i) \times C_j\right) \;\; \forall j \in \mathcal{O} \tag{3.50}$$

Any other combination $\bar{c}$ rather than $\bar{c}_1 = \mathcal{O}$ causes a lateness shorter than Λ_i^q. $\bar{c}_l$ causes at most a lateness of

$$\Lambda_i^q - \sum_{j \in \mathcal{O}, j \notin \bar{c}} wl_{j,over}^q \tag{3.51}$$

In Figure 3.10, if τ_3 meets its deadline, then the execution of the second instance of τ_1 will not interfere with the execution of τ_3. Let Γ_i^q denote the interfering workload that appear after $\delta_i^-(q) + D_i$. This workload will not interfere any more with the τ_i's instance if it meets its deadline. Hence, we can say that the lateness caused by $\bar{c}$ is refined to:

$$\Lambda_i^q - \Gamma_i^q - \sum_{j \in \mathcal{O}, j \notin \bar{c}} wl_{j,over}^q \tag{3.52}$$

q meets its deadline if its lateness is $\leqslant 0$. Thus, $\bar{c}$ is a schedulable combination if

$$\forall q \in [1, Q_i] : \sum_{j \in \mathcal{O}, j \notin \bar{c}} wl_{j,over}^q \geqslant \Lambda_i^q - \Gamma_i^q \tag{3.53}$$

We compute Γ_i^q as follows:

$$\Gamma_i^q := \sum_{j \in \mathcal{Z} \setminus \{i\}} \max\Big(\alpha_j - \beta_j, 0\Big) \tag{3.54}$$

Where α_j denotes the workload of τ_j up to the completion time of q-th instance ($B_i^+(q)$), and β_j denotes the workload of τ_j up to the relative deadline ($\delta_i^-(q) + D_i$). From Equation 3.46, we can compute α_j directly:

$$\alpha_j = \min\left(\left\lceil \frac{q \,.\, C_i}{\theta_i} \right\rceil \times \theta_j \;,\; \eta_j^+(B_i^+(q)) \times C_j\right) \tag{3.55}$$

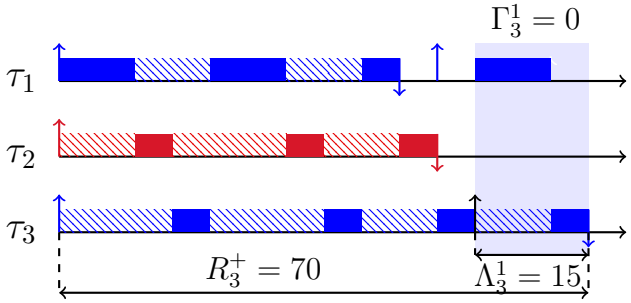

Figure 3.11: A corner case shows that the schedulability condition is sufficient: $\Gamma_3^1 = 0$.

Computing β_j is not direct like α_j. To define a safe and conservative condition, we should not over-approximate Γ_i^q, rather, it is safe when it is under-approximated. In Figure 3.10, τ_1 has $\frac{C_1}{\theta_1} \notin \mathbb{N}$. The figure illustrates that $\alpha_1 = \min\{40, 50\} = 40$ which means that the workload before D_i is bounded by $\lceil \frac{C_j}{\theta_j} \rceil \times \theta_j$ and not C_j. To safely approximate Γ_i^q we over-approximate β_j to cover the case when $\frac{C_j}{\theta_j} \notin \mathbb{N}$:

$$\beta_j = \min\left(\left\lceil \frac{q \;.\; C_i}{\theta_i} \right\rceil \times \theta_j \;, \left\lceil \frac{\eta_j^+(\delta_i^-(q) + D_i) \times C_j}{\theta_j} \right\rceil \times \theta_j \right) \tag{3.56}$$

Theorem 3.5. *A combination $\bar{c}$ is schedulable if (sufficient condition):*

$$\forall q \in [1, Q_i], R_i^q > D_i : \sum_{j \notin \bar{c}} wl_{j,over}^q \geqslant \Lambda_i^q - \Gamma_i^q \tag{3.57}$$

Proof. When the lateness of q w.r.t. $\bar{c}$ is $\leqslant 0$, then q meets its deadline. We showed that $\Lambda_i^q - \Gamma_i^q - \sum_{j \in \mathcal{O}, j \notin \bar{c}} wl_{j,over}^q$ is an over-approximation on the lateness of q w.r.t. $\bar{c}$. Γ_i^q is under-approximated as Figure 3.11 shows. $\Gamma_3^1 = 0$ although the first time slot of the second instance of τ_1 will never interfere with the execution of τ_3 when it meets its deadline. There might be a case in which q meets its deadline w.r.t. a combination $\bar{c}'$ although $\sum_{j \notin \bar{c}'} wl_{j,over}^q < \Lambda_i^q - \Gamma_i^q$.

When all τ_i instances within BW_i^+ meet their deadline, then τ_i is guaranteed to be schedulable w.r.t. $\bar{c}$.

□

Consequently, the following set contains all unschedulable combinations w.r.t τ_i:

$$\tilde{\mathcal{C}}_i := \{\bar{c} \mid \forall q \in [1, Q_i], R_i^q > D_i : \sum_{j \notin \bar{c}} wl_{j,over}^q < \Lambda_i^q - \Gamma_i^q\} \tag{3.58}$$

Compute $\Omega_k^{j \to i}$. The last step to compute a DMM using TWCA is computing an upper bound $\Omega_k^{j \to i}$ on the number of overload instances that may impact the execution of any instance of the k-sequence.

Lemma 3.10.

$$\forall j \in \mathcal{O} : \Omega_k^{j \to i} := \eta_j^+(|BW_i^+| + \delta_i^+(k) + R_i^+) \tag{3.59}$$

Proof. The idea behind this lemma is similar to Lemma 3.3. We should carefully bound the window of k-sequence by considering the maximum distance between k instances $\delta_i^+(k)$ and sufficient time windows before and after it during which the execution of the first and the last instances of the k-sequence might be impacted.

$|BW_i^+|$: any overload instance interferes with the busy-window during which the first instance of the k-sequence is activated, then it may impact its execution.

R_i^+: any overload instance that may be activated within the response time of the last instance of the k-sequence, may impact its execution. □

Compute DMM. We can now formulate our problem of computing a DMM of a given task $\tau_i \in \mathcal{T}$ as an ILP.

Theorem 3.6. *The ILP in the following equation is a DMM for a given task $\tau_i \in \mathcal{T}$ when WRR scheduling policy is considered.*

$$dmm_i(k) := N_i \ . \ \max\Big\{ \sum_{\bar{c} \in \tilde{\mathcal{C}}_i} x_{\bar{c}} \ : \ \sum_{\bar{c}: j \in \bar{c} \in \tilde{\mathcal{C}}_i} x_{\bar{c}} \leqslant \Omega_k^{j \to i} \ \forall j \in \mathcal{O}, \ x_{\bar{c}} \in \mathbb{N} \ \forall \bar{c} \in \tilde{\mathcal{C}}_i \Big\} \tag{3.60}$$

Proof. Similar argumentation as in Theorem 3.3. □

3.4 DMM for EDF Scheduling Policy

EDF scheduling has been proved to be optimal [30] in the sense of feasibility under certain conditions. This means that if there exists a feasible schedule for a task set, then EDF scheduling is able to find a feasible schedule as well. Also, EDF is able to schedule a task set with a utilization up to $U = 1$. However, EDF is less predictable than fixed priority scheduling, especially for systems may miss their deadline. EDF has another drawback related to handling overload: it is possible

in the extreme case that the activation of new task, such as exception handling routine, causes all tasks to miss their deadline. This phenomenon is known as the *domino effect*. Handling overload conditions under EDF has been addressed by plenty of papers [19, 61, 70] where the overload sources are unexpected or unknown. TWCA applies to systems for which the overload is expected and modelled by arrival curves, see Definition 2.7. TWCA aims to compute DMMs for tasks that are vulnerable to deadline-misses due to transient overload. Hence, establishing an on-line planning [96, 11] to dynamically handle overload conditions is not in the scope of this section.

In this section, we present a method for computing a DMM for independent tasks under EDF scheduling policy. This section follows the same structure as the sections 3.2 and 3.3: presentation of the EDF scheduling policy, worst-case response time analysis and DMM computation.

3.4.1 EDF scheduling policy

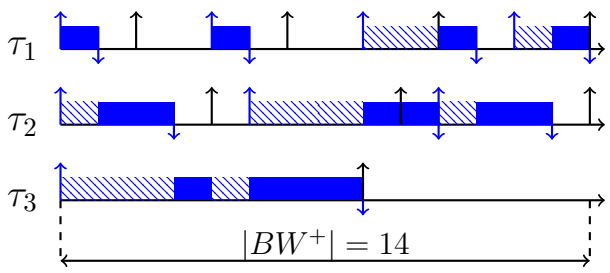

Figure 3.12: The longest busy-window under EDF scheduling policy. The black upward arrow indicates D_i.

EDF is a dynamic priority scheduling algorithm, i.e., a task priority is changed regularly upon the strategy followed in the scheduling policy. In EDF, the task with the closest absolute deadline gets the resource. We consider a preemptive EDF scheduling, so the execution of a task may be interrupted by the instance of another task with an earlier absolute deadline. In addition, EDF assigns the highest priority to the instance with a missed deadline to complete execution. Therefore, it is a deadline-miss agnostic scheduler, i.e., it schedules tasks and lets them run to completion even if they have missed their deadline. Note that offset is not considered in this section.

3.4.2 Worst-case response time analysis

It is sufficient for hard real-time systems to test the schedulability with no need to computing the response time. In this case, a sufficient and necessary schedulability test can be satisfactory. In this work we are interested in computing the response time and in the schedulability test as well. However, finding the worst-case response time of a task is not trivial when EDF scheduling is considered. In this section, we show how to compute the worst-case response time R_i^+ for a given task τ_i and the length of the longest busy-window [110].

Note that for EDF, the busy-window is not defined w.r.t. a specific task as we did for FPP, FPNP, and WRR, rather, it is defined w.r.t. the task set. Therefore, we use BW^+ to denote the busy-window. Figure 3.12 shows BW^+.

Under EDF, the worst-case response time is not found necessary in the longest busy-window.

Lemma 3.11 ([110])**.** *The worst-case response time of a task τ_i is found in a busy-window in which all tasks other than τ_i are released synchronously at the beginning of the busy-window and then at their maximum rate.*

Proof. This lemma is lemma 4.1 in [110]. The key argument to prove this lemma is that if all instances of tasks different from τ_i are "shifted left" such that they are released synchronously at the beginning of the busy-window, the workload of these instance cannot diminish. □

To compute the response time for an instance ℓ of τ_i activated at $\mathfrak{a}$ with absolute deadline d_i^ℓ, it is sufficient to be aware about a part of the busy-window in which only task instances with absolute deadline smaller than or equal to d_i^ℓ execute. This part of the busy-window relative to the absolute deadline d_i^ℓ is named *deadline-d busy-window* and $L_i(\mathfrak{a})$ denotes its length.

We are looking for an instance ℓ activated at time $\mathfrak{a} \geqslant 0$ such that R_i^ℓ is the worst-case response time: $R_i^+ = R_i^\ell$. The first instance of τ_i in the busy-window is activated at $\alpha_i^0(\mathfrak{a})$:

$$\alpha_i^0(\mathfrak{a}) = \mathfrak{a} - \delta_i^-(\eta_i^+(\mathfrak{a})) \tag{3.61}$$

BW^+ is the maximum length of any busy-window, therefore, the significant values of $\mathfrak{a}$ are in the interval $[0, BW^+ - C_i[$. Furthermore, in [110] author claims that it is not difficult to see that the local maxima of $L_i(\mathfrak{a})$ are found for those values of $\mathfrak{a}$, such that in the arrival pattern there is at least an instant of a task different from τ_i with deadline equal to d_i^ℓ, or in the longest busy-window, i.e. $\alpha_i^0(\mathfrak{a}) = 0$. Then it is sufficient to say:

$$R_i^+ = \max_{\substack{\mathfrak{a}=\mathfrak{d}-D_i \\ \mathfrak{d}\in\mathbb{D}}} \{R_i^\ell\} \tag{3.62}$$

where R_i^ℓ is the response time of an instance ℓ activated at $\mathfrak{a}$, and $\mathbb{D}$ is the set of absolute deadlines of other tasks when they are released synchronously at $t = 0$:

$$\mathbb{D} = \cup_{j \in \mathcal{Z}} \{d^l | d^l = \delta_j^-(l) + D_j \wedge d^l < |BW^+|, l \in \mathbb{N}^*\} \tag{3.63}$$

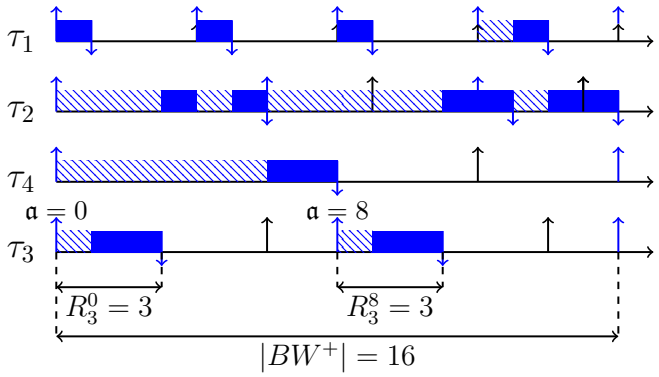

Figure 3.13: The response time of the instances of τ_3 that are activated at $\mathfrak{a} = 0$ and $\mathfrak{a} = 8$. See Example 3.2 that is based on [110].

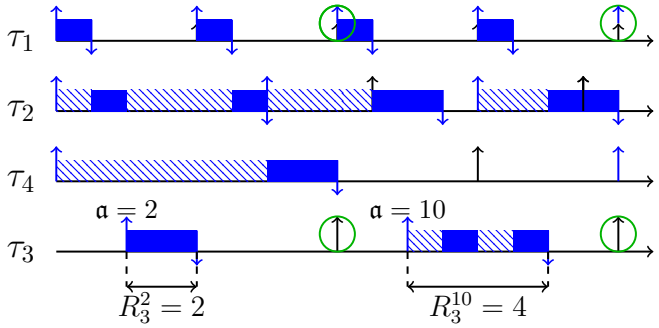

Figure 3.14: The response time of the instances of τ_3 that are activated at $\mathfrak{a} = 2$ and $\mathfrak{a} = 10$.

Example 3.2. *Let us consider the task set shown in Table 3.1. We want to compute the worst-case response time of* τ_3*. In this system* $|BW^+| = 16$ *as Figure 3.13 shows. The set* $\mathbb{D}$ *is:* $\mathbb{D} = \{6, 8, 9, 12, 14, 15, 16\}$*. The equivalent values of* $\mathfrak{a}$ *are shown in Table 3.1. The worst-case response time is highlighted in red. Figures 3.13, 3.14, 3.15, 3.16 and 3.17 illustrate the response time of instances that are activated at* $\mathfrak{a}$*.*

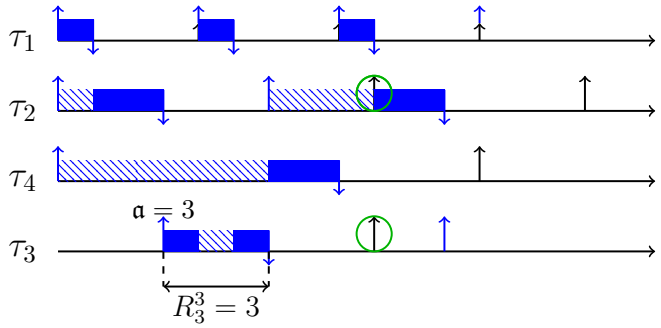

Figure 3.15: The response time of the instance of τ_3 that is activated at $\mathfrak{a} = 3$.

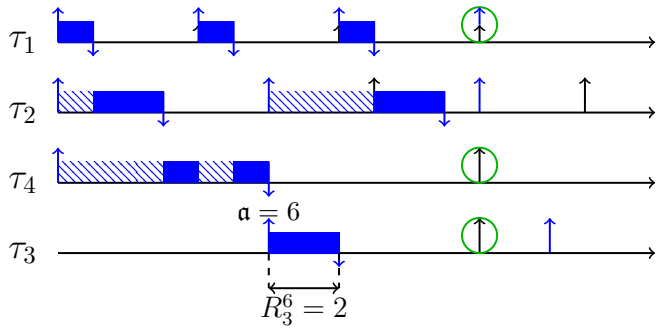

Figure 3.16: The response time of the instance of τ_3 that is activated at $\mathfrak{a} = 6$.

3.4.2.1 Algorithm to compute worst-case response time

We start with bounding the length of the busy-window $|BW^+|$ iteratively starting with $|BW^+| = \sum_{j \in \mathcal{Z}} C_j$ using the following equation:

$$|BW^+| := \sum_{j \in \mathcal{Z}} \eta_j^+(|BW^+|) \; . \; C_j \tag{3.64}$$

After bounding $|BW^+|$, we can compute $\mathbb{D}$ which represents the set of candidates to bound the worst-case response time, Equation 3.63.

The next step is to bound the contribution of tasks in the deadline busy-window $L_i(\mathfrak{a})$. Assume that the beginning of the busy-window is at $t_0 = 0$. Up to time t, $\eta_j^+(\Delta = t - 0)$ instances of τ_j will have been released for $j \neq i$ and there will be no more than $\tilde{\eta}_j^+(\mathfrak{a} + D_i - D_j)$ instances contribute in $L_i(\mathfrak{a})$.

$$W_i(\mathfrak{a}, t) = \sum_{\substack{j \neq i \\ D_j \leqslant \mathfrak{a} + D_i}} \min\{\eta_j^+(t), \tilde{\eta}_j^+(\mathfrak{a} + D_i - D_j)\} \; . \; C_j + \lambda_i(\mathfrak{a}, t) \; . \; C_i \tag{3.65}$$

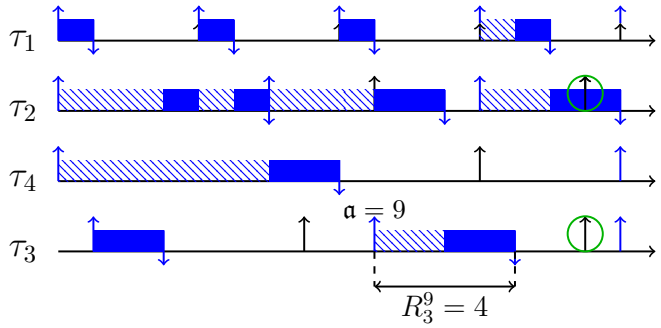

Figure 3.17: The response time of the instance of τ_3 that is activated at $\mathfrak{a} = 9$.

task	C_i	D_i	T_i
τ_1	1	4	4
τ_2	2	9	6
τ_3	2	6	8
τ_4	2	12	16

$\mathfrak{a}$	R_3^ℓ
0	3
2	2
3	3
6	2
8	3
9	4
10	4

Table 3.1: The worst-case response time analysis of τ_3, based on [110].

Where $\lambda_i(\mathfrak{a}, t)$ is equivalent to $\delta_i(\mathfrak{a}, t)$ in [110]:

$$\lambda_i(\mathfrak{a}, t) = \begin{cases} \min\{\eta_i^+(t - \alpha_i^0(\mathfrak{a})), \tilde{\eta}_j^+(\mathfrak{a})\} & t > \alpha_i^0(\mathfrak{a}), \\ 0 & otherwise. \end{cases} \tag{3.66}$$

Then, the deadline busy-window $L_i(\mathfrak{a})$ is bounded as follows:

$$\begin{cases} L_i^{(0)}(\mathfrak{a}) = \sum_{\substack{j \neq i \\ D_j \leqslant \mathfrak{a} + D_i}} C_j + I_{\{\alpha_i^0(\mathfrak{a})=0\}} \cdot C_i \\ L_i^{(m+1)}(\mathfrak{a}) = W_i(\mathfrak{a}, L_i^{(m)}(\mathfrak{a})) \end{cases} \tag{3.67}$$

Where

$$I_{\{\alpha_i^0(\mathfrak{a})=0\}} = \begin{cases} 1 & \alpha_i^0(\mathfrak{a}) = 0, \\ 0 & otherwise. \end{cases} \tag{3.68}$$

Note that Equation 3.67 is equivalent to Equation 3 in [110].

The response time of instance ℓ that is activated at $\mathfrak{a}$ is then:

$$R_i^\ell = \max\{L_i(\mathfrak{a}) - \mathfrak{a}, C_i\} \tag{3.69}$$

The computation of the worst-case response time of a given task τ_i is shown in Algorithm 2.

Algorithm 2: Compute the worst-case response time of τ_i under EDF

```
Compute |BW+| ;
for 𝔡 ∈ 𝔻 do
    𝔞 = 𝔡 − D_i;
    t_0 = 0;
    α_i^0(𝔞) = 𝔞 − δ_i^−(η_i^+(𝔞));
    Compute L_i(𝔞) (3.67);
    Compute R_i^ℓ (3.69);
R_i^+ = max{R_i^ℓ}
```

3.4.2.2 Sufficient and necessary schedulability test

For EDF scheduling, a sufficient and necessary schedulability test has been developed based on the *demand bound function.*

Definition 3.3. *Demand bound function dbf_i is defined as follows:*

$$dbf_i(t) := \tilde{\eta}_i^+(t - D_i) \; . \; C_i \tag{3.70}$$

Theorem 3.7. *A task set $\mathcal{Z}$ is schedulable if and only if:*

$$\forall t \in \mathbb{D} : \sum_{\substack{\forall i \in \mathcal{Z} \\ D_i \leqslant t}} dbf_i(t) \leqslant t \tag{3.71}$$

Proof. It has been proven in Theorem 6 in [69] and Theorem 3.1 in [110] that if a deadline-miss can occur in a busy-window, then a deadline-miss occurs in the longest busy-window. Therefore, it is sufficient to test the schedulability in the longest busy-window.

At each point of time $t \in \mathbb{D}$ if the demanded workload to be processed up to t is less than t, then the absolute deadline at t will be guaranteed to be met. Thus, it is necessary to test the schedulability $\forall t \in \mathbb{D}$. □

3.4.3 Deadline miss model computation

We now show how TWCA can be adapted to compute DMMs for real-time tasks that are scheduled with EDF.

To compute a DMM for a given task τ_i using TWCA:

1. we compute an upper bound N_i on the number of deadlines that τ_i may miss within one busy-window;

2. we propose a criterion to compute efficiently the unschedulable combinations , see Definition 3.2;

3. we compute an upper bound $\Omega_k^{j\to i}$ on the number of overload instances of τ_j that may interfere with any busy-window containing instances of the k-sequence.

Compute N_i. Satisfying Property 3.2 implies that the impact of overload instances is enclosed in the busy-window during which they have been executed. Hence, the number of deadlines that τ_i may miss within one busy-window due to one or more overload instances can be bounded.

Lemma 3.12. *For a given task τ_i, let $\mathbb{BW}_i$ indicate the set of busy-windows that each of which satisfies: all tasks but τ_i are activated synchronously at the beginning of the busy-window and then at their maximum rate; and the first instance of τ_i in the busy-window is activated at $\mathfrak{a}$ where $\mathfrak{a} = \mathfrak{d} - D_i : \mathfrak{d} \in \mathbb{D}$. Also, $\forall b \in \mathbb{BW}_i$ let n_b indicate the number of deadline-misses within the busy-window b. Let*

$$N_i = \max_{\forall b \in \mathbb{BW}_i} \{n_b\} \tag{3.72}$$

Then N_i is an upper bound on the number of deadlines that τ_i may miss within one busy-window.

Proof. Unlike FPP, FPNP and WRR, there is more than one busy-window candidate in EDF during which response times of all instances of τ_i upper bound response times of instances in other busy-windows.

It has been proven in [110] that within a busy-window in which all tasks but τ_i are activated synchronously at the beginning of the busy-window and then at their maximum rate, the workload of all other tasks cannot diminish and the response time of instances of τ_i within this busy-window can only increase. A certain distribution of the interfering workload will cause the maximum number of deadline-misses to τ_i within a busy-window. Such a busy-window is not necessary to be the longest one, in which $\mathfrak{a} = 0$. Therefore, all possible busy-window candidates have to be checked. □

Schedulability criterion. A task τ_i misses its deadline when it experiences interfering from unschedulable combinations, see Definition 3.2. To compute DMMs using TWCA, the set of unschedulable combinations has to be computed. In EDF scheduling policy, a sufficient and necessary schedulability test based on the demand bound function is used [110]. The demand bound function is defined in Definition 3.3, and the schedulability test is presented in Equation 3.71. Based on this schedulability test we compute the set of unschedulable combinations as follows:

Lemma 3.13. *A combination $\bar{c}$ is unschedulable if (necessary schedulability condition) the task set $\mathcal{T} \cup \bar{c}$ is not schedulable:*

$$\exists t \in \mathbb{D}_{\bar{c}} : \sum_{\substack{\forall j \in \mathcal{T} \cup \bar{c} \\ D_j \leqslant t}} dbf_j(t) > t \tag{3.73}$$

where

$$\mathbb{D}_{\bar{c}} = \cup_{j \in \mathcal{T} \cup \bar{c}} \{d^l | d^l = \delta_j^-(l) + D_i \wedge d^l < |BW^+|, l \in \mathbb{N}^*\}$$

Proof. The condition in Equation 3.73 implies that there is at least one instance of $\tau_j : \forall j \in \mathcal{T} \cup \bar{c}$ misses its deadline. However, it does not mean that the given task τ_i misses its deadline with the presence of $\bar{c}$. That is to say, the generated set of unschedulable combinations using the above condition is a superset of the set of unschedulable combinations w.r.t. τ_i.

□

The superset of unschedulable combinations w.r.t. τ_i is then:

$$\tilde{\mathcal{C}}_i := \{\bar{c} \mid \exists t \in \mathbb{D}_{\bar{c}} : \sum_{\substack{\forall i \in \mathcal{T} \cup \bar{c} \\ D_i \leqslant t}} dbf_i(t) > t\} \tag{3.74}$$

Note that for EDF scheduling $\forall i, j \in \mathcal{T} : \tilde{\mathcal{C}}_i = \tilde{\mathcal{C}}_j$ because $\tilde{\mathcal{C}}$ is computed regardless the considered task as explained in Lemma 3.13.

Compute $\Omega_k^{j \to i}$. Computing a DMM using TWCA requires computing an upper bound $\Omega_k^{j \to i}$ on the number of overload instances that may impact the execution of any instance of the k-sequence.

Lemma 3.14. *$\Omega_k^{j \to i}$ is bounded as follows:*

$$\forall j \in \mathcal{O} \ : \ \Omega_k^{j \to i} := \tilde{\eta}_j^+(|BW^+| + \delta_i^+(k) + \max\{D_i - D_j, 0\}) \tag{3.75}$$

Proof. Figure 3.18 illustrates $\Omega_k^{j \to i}$. We know that $\tilde{\eta}_j^+(\Delta)$ returns the maximum number of instances that may occur within a closed time interval. Just like for FPP and WRR, Lemma 3.3 and 3.10 respectively, we should carefully bound the window of k-sequence by considering the maximum distance between k instances and sufficient time windows before and after it during which the execution of the first and the last instances of the k-sequence might be impacted.

$\delta_i^+(k)$: the longest closed time duration that contains k consecutive instances is bounded by $\delta_i^+(k)$. An overload instance that occurs during it may have an impact on the response times of the k-sequence.

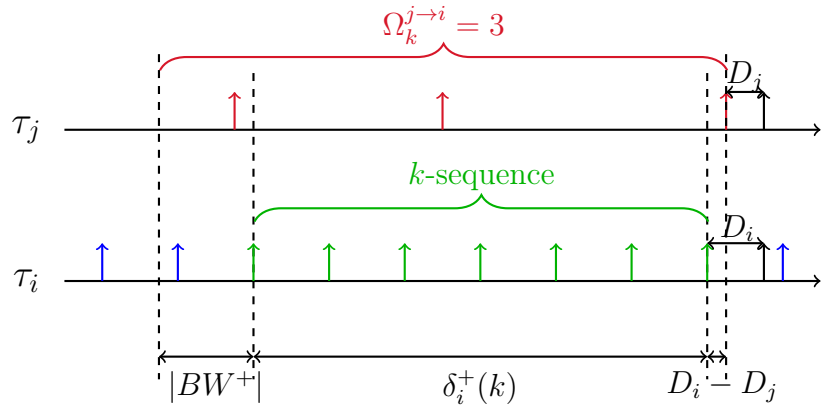

Figure 3.18: $\Omega_k^{j\to i}$ under EDF scheduling policy. The black upward arrow indicates D.

$|BW^+|$: An overload instance will have no impact on the first instance of the k-sequence unless they belong to the same busy-window. $|BW^+|$ is proven to be the maximum length of a busy-window.

$\max\{D_i - D_j, 0\}$: Any overload instance with an absolute deadline beyond the absolute deadline of the k-th instance has no impact on it. In Figure 3.18, the third instance of τ_j is activated after the k-th instance of τ_i by $D_i - D_j$ and it may therefore interfere with the execution of the k-th instance. □

Compute DMM. Each unschedulable combination may impact at most one busy-window causing at most N_i deadline-misses. The number of impacted busy-windows varies depending on the way we combine the available $\sum_{j\in\mathcal{O}} \Omega_k^{j\to i}$ overload instances in unschedulable combinations. To conservatively compute the DMM, we need to maximize the number of impacted busy-windows, thus, the following theorem:

Theorem 3.8. *We compute a DMM for any task τ_i using the ILP in the following equation.*

$$dmm_i(k) := N_i \ . \ \max\Big\{ \sum_{\bar{c}\in\tilde{\mathcal{C}}_i} x_{\bar{c}} \ : \ \sum_{\bar{c}: j\in\bar{c}\in\tilde{\mathcal{C}}_i} x_{\bar{c}} \leqslant \Omega_k^{j\to i} \ \forall j \in \mathcal{O},\ x_{\bar{c}} \in \mathbb{N} \ \forall \bar{c} \in \tilde{\mathcal{C}}_i \Big\} \tag{3.76}$$

Proof. Similar argumentation as in Theorem 3.3. □

3.5 Related Work

In this section we address the research papers that are related to this chapter. The related work will be shown in two categories: papers that provide WHRT guarantees and papers that provide probabilistic guarantees.

In control engineering, a natural application domain of real-time computing beside signal processing and others, the period/deadline is an efficient interface between control & timing [128]. On the one hand, the deadline represents the control latency to guarantee stability. On the other hand, the shorter the deadline, the better performance and robustness can be guaranteed [23]. Therefore, a better compromise can be achieved between control performance and timing performance by accepting few deadline-misses. Many research papers have been dedicated to optimizing the establishing of deadlines [108, 73, 23, 41]. It has been shown that the hard deadline represents the maximum allowable control latency to guarantee stability and it is typically several times larger than the sampling period [108]. The deadline that is set to be equal to the period, Magalhães et al. in [73] called it the performance deadline, can be missed safely as along as the hard deadline is guaranteed. From the timing viewpoint, there is a need for a less-than-worst-case analysis that accepts deadline-misses as long as they are tolerable by the controller.

3.5.1 WHRT analysis

Hamdaoui et al. coined in [47] the notation (m, k)-firm to define the timing constraints of real-time tasks. A real-time task can tolerate at most m deadline-misses in a k-sequence. Ramanathan in [97], and Quan and Hu in [88] exploited the notation (m, k)-firm to manage overload in control applications by taking the advantage of tolerating deadline-misses. Their system model considers periodic tasks with FPP scheduling.

The system in which tasks follow the (m, k)-firm model was formally defined by Bernat et al. in [7] as *weakly-hard real-time systems*. Bernat et al. defined a WHRT analysis to provide bounds on the maximum number of deadline-misses that a task may miss within a given time window. The system model considers tasks that are scheduled with a known initial offset. The proposed analysis computes the number of deadline-misses in a time-window of k consecutive instances along the system hyper-period (i.e., the least common multiple of all task periods), therefore, [7] is only defined for periodic tasks.

Sun et al. presented in [113] a WHRT schedulability analysis that bounds the maximum number of deadline-misses within a time window of k consecutive instances using an MILP. The system model considers *offset-free* tasks. The MILP

checks all possible scenarios within a time-window of k consecutive instances. Therefore, [113] can provide tighter bounds than [7] but with higher complexity. [113] is limited to periodic tasks; and does not scale beyond 20 tasks and $k > 10$ [80].

In [85], Pazzaglia et al. researched the performance cost of deadline-misses in control systems. Their contribution can be considered as the first analysis to extract safe (m, k) constraints for control systems.

Kumar et al. proposed an analysis in [62] in the context of RTC [24]. In that paper, rare events represent the possible deviation from the nominal timing model (corresponding to the typical model in TWCA [91]). The presented analysis computes the settling time, i.e., the longest time window after the rare event until the system returns to normal. In addition, the overshoot during the settling time quantifies how many deadline-misses may then occur. The main difference with TWCA is that [62] considers only a single temporal overload.

Establishing an on-line scheduling framework for WHRT systems has been addressed in [8]. Bernat and Cayssials proposed an on-line scheduling framework called Bi-Modal Scheduler that is characterized by two modes of operation. WHRT constraints are guaranteed to be satisfied by switching, whenever necessary, from a normal mode to a panic mode for which schedulability tests exist that guarantee the constraints on the allowed number of deadline-misses. Similarly, [67] proposed an on-line (m, k)-firm enforcement policy for control systems with nonpreemptive EDF scheduling. In this thesis, we aim to compute DMMs for tasks that are vulnerable to deadline-misses due to transient overload. Hence, establishing an on-line scheduling is out of the scope of this thesis.

3.5.2 Probabilistic analysis

A *stochastic analysis* of periodic real-time systems is presented in [31, 71]. The analysis follows the probabilistic real-time model in which at least one parameter is described by a random variable. The analysis is applicable to uniprocessor systems under FPP or EDF. The probability that a task misses its deadline in an infinite window size is computed. The stochastic model has been further developed in [58] for the case of dependent variables.

Carnevali et al. considered in [20] periodic tasks sharing a uniprocessor under FPNP scheduling policy. In that paper, they presented a *probabilistic deadline-miss analysis*. The analysis supports the derivation of the probability of missing a deadline within time t. In this work, instances are discarded as soon as their deadline is missed. The analysis requires solving Markov renewal equations, which makes the analysis computationally extremely expensive.

In [103], Santinnelli and Cucu-Grosjean presented a *probabilistic calculus*. The analysis is developed in terms of sufficient probabilistic schedulability conditions for task systems with either FPP or EDF scheduling policies. That work follows CPA, therefore, it is applicable to DRTSs. The probability of missing a deadline of a periodic or non-periodic task in an infinite window size can be derived from the proposed analysis.

Probabilistic real-time approaches [31, 71, 20, 103], at least in principle, could be used to compute the probability of missing m deadlines out of k consecutive instances. This analysis is likely to be computationally extremely expensive when applied to the (m, k) analysis. However, probabilistic deadline guarantee is not sufficient because information about the sequence of occurrence is missing in probabilistic bounds. Such information has quite an impact on the system performance for real-time systems such as automatic control systems [84]. Instead, a precise bound on the distribution of the system met and missed deadlines in time is necessary and this can be done using (m, k)-firm model [67].

In [120] some predefined tasks are allowed to miss their deadlines occasionally in uncertain or faulty execution conditions due to soft errors. Real-time guarantees were computed to determine if the system can provide full timing guarantees, i.e., all tasks meet their deadline, or limited timing guarantees, i.e., a subset of tasks met their deadline, and to determine the maximum interval length until the system will again provide full timing guarantees. The system model considers independent sporadic tasks in a uniprocessor system under a fixed-priority scheduling policy. With the same system model used in [120] and based on probabilistic WCETs, Chen et al. proposed an analysis in [25] to calculate the probability of missing m consecutive deadlines of a task in faulty execution conditions due to soft errors. Later, Von Der Brüggen et al. presented in [121] two directions to evaluate the probability of missing a deadline. One approach is based on analytical upper bounds that can be efficiently computed in polynomial time at the price of precision loss based on Hoeffding's and Bernstein's inequalities. Another approach convolutes the probability efficiently over multinomial distributions.

In the meantime of writing this thesis two interesting papers have been published [38, 2], which build on top of the results presented in this manuscript. Ahrendts et al. in [2] proposed an analysis to compute end-to-end WHRT guarantees in the form of a DMM for switched network using TWCA. That analysis is the first work that addresses the schedulability problem of distributed WHRT systems. In [38], Fradet et al. presented a formal proof of TWCA using the Coq proof assistant [87]. That work is the first formal proof of an analysis for computing weakly-hard guarantees.

3.6 Summary

Throughout this chapter, we have an analysis presented to compute WHRT guarantees in the form of a DMM for temporarily overloaded uniprocessor systems scheduled with FPP, FPNP, WRR or EDF scheduling policies using TWCA. The DMM computation was formulated as an ILP that is independent of the scheduling policy. The coefficients of the objective function and the constraints are computed according to the considered scheduler. An LP relaxation was proposed to improve the solution efficiency.

In the next chapter, we discuss the efficiency, scalability, and complexity of the proposed analysis as well as the pessimism of the computed DMMs and general properties on DMMs.

4 ANALYTICAL AND EXPERIMENTAL EVALUATION OF DMMs

This chapter discusses properties of DMMs and the efficiency of the proposed analysis through intensive and detailed experiments.

4.1 Analytical evaluation of DMMs

In this section we underline some properties of $dmm_i(k)$, then we analytically evaluate the complexity of computing a DMM using the proposed analysis. We address the sources of pessimism in computing DMMs as well, and we show the impact of the schedulability criterion, which is used to classify the combinations into schedulable and unschedulable, on the quality of computed DMMs.

Let us recall here the formula for computing DMMs given in Section 3.2.3.

$$dmm_i(k) = N_i \ . \ \max\Big\{ \sum_{\bar{c}\in\tilde{\mathcal{C}}_i} x_{\bar{c}} \ : \ \sum_{\bar{c}: j\in\bar{c}\in\tilde{\mathcal{C}}_i} x_{\bar{c}} \leqslant \Omega_k^{j\to i} \ \forall j \in \mathcal{O}, \, x_{\bar{c}} \in \mathbb{N} \ \forall \bar{c} \in \tilde{\mathcal{C}}_i \Big\} \tag{4.1}$$

Note that it does not depend on the scheduling policy. What changes w.r.t. to the considered scheduling policy is how N_i, $\tilde{\mathcal{C}}_i$, and $\Omega_k^{j\to i}$ are computed.

Property 4.1. *dmm_i is a non-decreasing function:*

$$\forall k_1, k_2 \in \mathbb{N}^+ : k_1 < k_2 \to dmm_i(k_1) \leqslant dmm_i(k_2) \tag{4.2}$$

Proof. In the computation of $dmm_i(k)$, N_i is a constant and independent of k. Also, whether a combination $\bar{c}$ is schedulable or not is independent of k. However, $x_{\bar{c}}$ is bounded by $\Omega_k^{j\to i}$, which is a function of k as Lemma 3.3 shows.

If $k_1 < k_2$, then $\Omega_{k_1}^{j\to i} \leqslant \Omega_{k_2}^{j\to i}$. Therefore, the packing[1] of $\Omega_{k_1}^{j\to i}$ instances into the time window of k_1 is valid for the time window of k_2 and more overload instances can only increase the number of impacted busy-windows. Thus, $dmm_i(k_1) \leqslant dmm_i(k_2)$. □

[1]Remember that the problem of computing DMMs is a multidimensional knapsack problem.

Figure 4.1 illustrates, dmm_i is a step function with a step size = N_i.

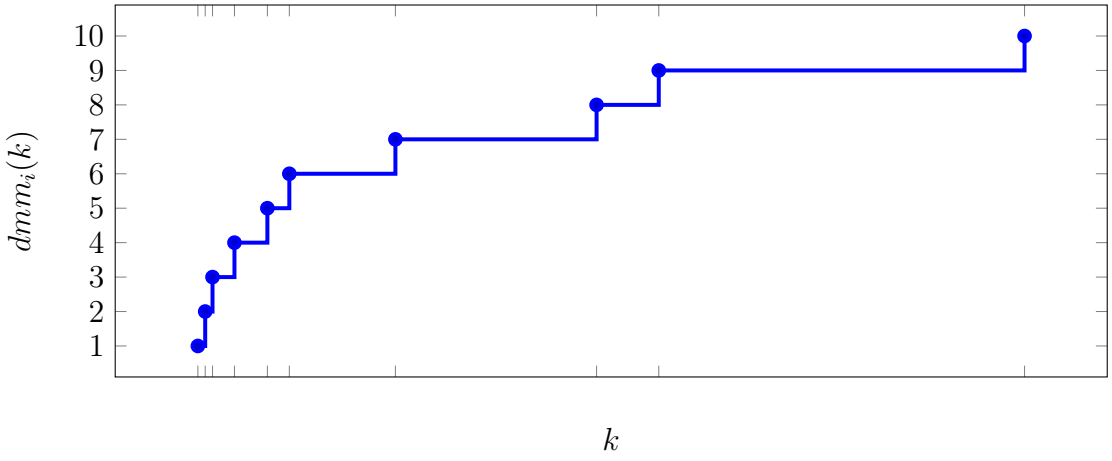

Figure 4.1: $dmm_i(k)$ for $k = [1, 3, 5, 11, 20, 26, 55, 110, 127, 227]$ where $N_i = 1$, depending on the synthetic example used in [50, 125].

4.1.1 Complexity

The general scheduling problem, i.e., mapping n tasks to m resources such that all timing constraints are met, has been proven to have the complexity NP-Complete [40].

The ILP presented in Equation 4.1 to compute $dmm_i(k)$ is a multidimensional knapsack problem as has been shown in Section 3.2.3. Finding an approximate algorithm, i.e., an algorithm that compute suboptimal solution, for a multidimensional knapsack problem is NP-Hard [74]. However, Algorithm 1 has the complexity of P.

4.1.2 Sources of pessimism

Computing $dmm_i(k)$ using the proposed analysis has three main sources of pessimism:

1) N_i is only an upper bound on the number of deadlines that τ_i may miss within one busy-window. Not every unschedulable combination causes the same number of deadline-misses. Figure 4.2 illustrates this case. While the combination $\bar{c}_1 = \{\tau_1, \tau_2\}$ causes two deadline-misses and therefore $N_i = 2$, $\bar{c}_2 = \{\tau_2\}$ cannot cause more than one deadline-miss. When $N_i = 1$, there is no pessimism related to N_i.

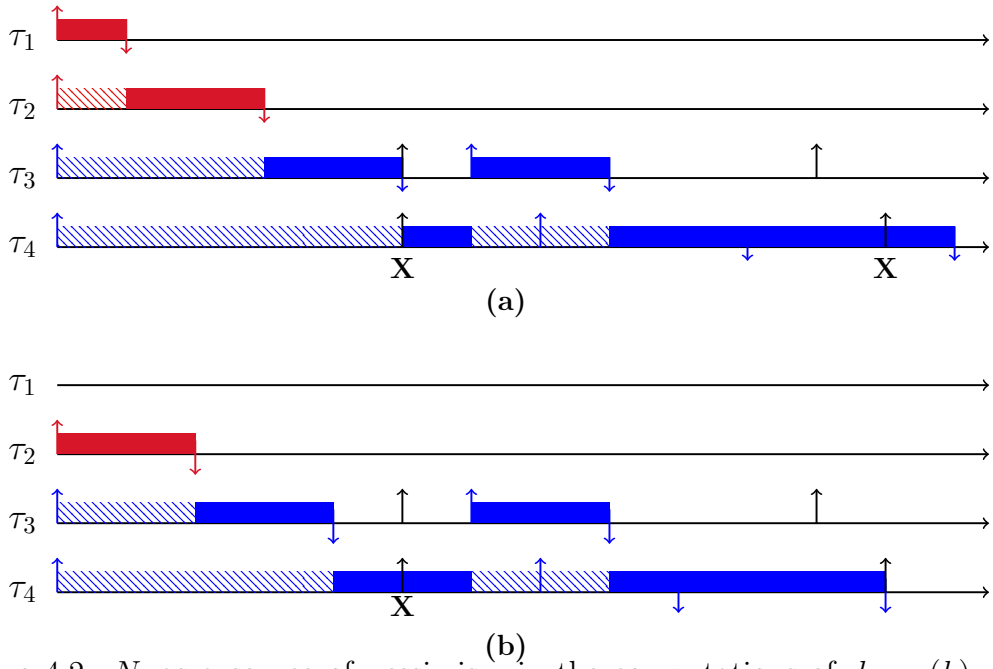

Figure 4.2: N_i as a source of pessimism in the computations of $dmm_i(k)$. The black upward arrows indicate the deadline. **X** indicates a deadline-miss.

2) TWCA assumes that every busy-window within the time window of the k-sequence experiences the maximum interference from all tasks $\tau_i \in \mathcal{T}$. This may not be possible given the activation models of tasks in $\mathcal{T}$ within the time window of the k-sequence. Consequently, the maximum number of impacted busy-windows are over-approximated.

Figure 4.3 shows a scenario in which τ_4 meets its deadline even with the presence of the *unschedulable combinations* $\bar{c}_2 = \{\tau_2\}$. Over a k-sequence, two busy-windows that both experience $\bar{c}_2$ cannot both miss deadlines because they cannot both be interfered by τ_3. Compare Figure 4.2.b with Figure 4.3 to observe the effect of τ_3.

3) When a sporadic task τ_j has more than one instance within the longest busy-window. Let ω_j denotes the number of instances of τ_j within BW_i^+. In a combination, each task τ_j has a Boolean representation: either ω_j instances are activated as early as possible according to the activation model of τ_j when $\tau_j \in \bar{c}$ or none of them when $\tau_j \notin \bar{c}$. Note that experiments show that this limitation rarely incurs pessimism.

4.1.2.1 Schedulability criterion impact

When the schedulability criterion is a sufficient and necessary condition for schedulability, it does not introduce any additional pessimism to the computation of

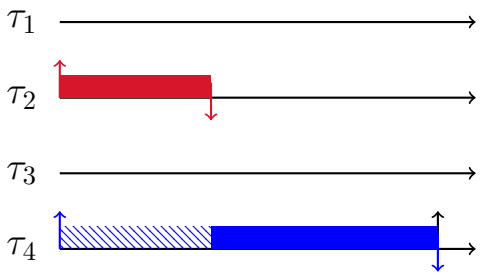

Figure 4.3: Second source of pessimism in the computation of $dmm_i(k)$. When τ_3 does not interfere with τ_4, then the combination $\bar{c}_2 = \{\tau_2\}$ is not unschedulable any more.

DMMs because it returns the exact set of unschedulable combinations. However, when the schedulability criterion is only a sufficient condition then it may lead to an over-approximation of $dmm_i(k)$ because it returns a superset of unschedulable combinations.

To clarify the impact of having only a sufficient schedulability criterion, consider the following system with 4 tasks $\tau_1, \tau_2, \tau_3, \tau_4$ where τ_1, τ_2, τ_3 are overload tasks. We want to compute $dmm_4(10)$ Assume that every combination $\bar{c}$ of two tasks or more is an unschedulable combination, and there is $\Omega_{10}^{1\to4} = \Omega_{10}^{2\to4} = \Omega_{10}^{3\to4} = 1$. There is only enough to impact one busy-window, therefore, $dmm_4(10) = N_i \times 1$. If the schedulability criterion is a sufficient condition such that the combination $\bar{c} = \{\tau_1\}$ is considered unschedulable, hence, $dmm_4(10) = N_i \times 2$.

4.2 Experiments

To evaluate extensively the proposed analysis, sets of synthetic test cases were developed considering a variety of systems. The experiments cover the FPP, FPNP, WRR and EDF scheduling policies, and they aim to study and illustrate the impact of different factors such as utilization, system size, etc. on the computation of $dmm_i(k)$.

Specifically, we investigated:

- How large the running time of our analysis is, and how it evolves as a function of k or of the number of unschedulable combinations.
- How the amount of sporadic overload impacts the provided guarantees.
- For the fixed-priority case, we additionally compared the preemptive and the non-preemptive case, as well as the impact of the priority assignment on the computed guarantees.

Note that the computed $dmm_i(k)$ using Equation 4.1 for a given task and a given k might be $dmm_i(k) \geqslant k$ due to the over-approximation in computing $dmm_i(k)$. To eliminate such non-reasonable results we compute $dmm_i(k)$ in the experiments as follows:

$$dmm_i(k) = \max\left\{k, N_i.\max\left\{\sum_{\bar{c}\in\tilde{\mathcal{C}}_i} x_{\bar{c}} : \sum_{\bar{c}:j\in\bar{c}\in\tilde{\mathcal{C}}_i} x_{\bar{c}} \leqslant \Omega_k^{j\to i}\ \forall j \in \mathcal{O}, x_{\bar{c}} \in \mathbb{N}\ \forall \bar{c} \in \tilde{\mathcal{C}}_i\right\}\right\} \tag{4.3}$$

In the computation of DMMs, we consider three cases:

- $k \leqslant Q_i$ where Q_i is the maximum number of instances of τ_i in one busy-window: $dmm_i(k) = \min\{k, N_i\}$.
- $n_s = 1$ where n_s is the number of sporadic overload tasks: $dmm_i(k) = \min\{k, N_i\ .\ \Omega_k^{s\to i}\}$.
- $k > Q_i \wedge n_s > 1$: we compute $dmm_i(k)$ using Equation 4.3.

Before presenting the obtained results, we detail the process that we followed to generate our test cases. This process follows the guidelines provided in [26].

4.2.1 Synthetic test case generation

Our synthetic test cases consist of a set of tasks with a worst-case execution time, a period (for periodic tasks), a minimum distance function (for sporadic tasks), and a relative deadline. Every task set $\mathcal{Z}$ consists of typical tasks that are chosen to be periodic in this experiments, and overload tasks, i.e., $\mathcal{Z} = \mathcal{T} \cup \mathcal{O}$. The following steps summarize how we generated them:

- We first choose the number of tasks $n = \#\{\mathcal{Z}\}$ and the number of overload tasks $n_s = \#\{\mathcal{O}\}$. We then decide on the system utilization to be shared among the tasks.
- UUnifast [9] is applied to assign a share of the system utilization to each task (sporadic or periodic, typical or overload).
- For typical tasks, we assign periods randomly chosen in a predefined set of harmonic values. Then, the worst-case execution time of typical tasks is computed as follows: $C_i = U_i * p_i$, where p_i denotes the period of the typical task τ_i. Note that $\delta_i^-(n) = (n-1)\ .\ p_i$.

- Generating minimum distance functions for sporadic tasks is not straightforward and there is no standard approach for this. In particular, they have to be super-additive, i.e., $\delta^-(a+b) \geqslant \delta^-(a) + \delta^-(b)$ for all a, b [53]. To achieve this, we depended on the definition of U_j of a sporadic task, which is defined as follows (see Definition 2.18):

$$U_j = \lim_{n \to \infty} \frac{(n-1) * C_j}{\delta_j^-(n)}$$

  We first randomly assign the worst-case execution time for each sporadic task $\tau_j \in \mathcal{O}$ such that $C_j \in [\min_{i \in \mathcal{T}}\{C_i\}, \max_{i \in \mathcal{T}}\{C_i\}]$. The justification of this choice is to not bias the activation model of τ_j toward having a high density of instances by selecting short execution times or low density by selecting long execution times. Then, we compute $\delta_j^-(N) = (N-1) * C_j/U_j$, with a sufficiently large N (for all our experiments $\delta_j^-(100)$ is much larger than the longest τ_i busy-window). We generated then a trace of N instances such that the first one is at 0 and the last one is at $\delta_j^-(N)$ and $N-2$ instances in between. We use pyCPA [33] to extract the minimum distance function from the generated trace (*model.TraceEventModel(trace)*).

To guarantee a wide variety of system models, we chose the various parameter values as follows.

- Number of tasks $n \in [3, 25]$ and number of sporadic overload tasks $n_s \in [1, 20]$. On the one hand, a system with $n < 3$ makes no sense to be analyzed. On the other hand, scheduling 25 tasks on one resource is realistic and acceptable, see the case study in Chapter 6.

- Total utilization is

$$U \in \{0.4, 0.5, 0.6, 0.7, 0.8\}.$$

  For many software design standards, e.g., ECSS Standard E-ST-40C for space software [34], the resource utilization should not exceed 0.75. Resources in the generated test cases therefore rang from average to maximum allowed utilization.

- The transient overload $U_\mathcal{O}$ has a share of

$$U_\mathcal{O}/U \in \{0.001, 0.01, 0.05, 0.1, 0.15, 0.2\}.$$

  The generated test cases range from being slightly overloaded to highly overloaded.

- Each typical task is assigned a relative deadline

$$D_i \in \{0.6, 0.8, 1, 1.2, 1.4\} \times \delta_i(2).$$

  Thus, the deadline can be constrained ($D_i \leqslant \delta_i^-(2)$), implicit ($D_i = \delta_i^-(2)$) or arbitrary ($D_i \geqslant \delta_i^-(2)$).

Note that we chose to have a range of values rather than just intervals to facilitate the study of parameters impact. These configurations apply to all experiments otherwise specified.

4.2.2 Fixed priority scheduling policy

We performed experiments for fixed-priority scheduling over 5000 synthetic test cases. We generated 5 sets of 1000 test cases. Every set of 1000 test cases has the same total utilization, e.g., $U = 0.4$. Priorities are assigned randomly to tasks. $\forall \tau_i \in \mathcal{T}$ we compute $dmm_i(k)$ for $k \in \{10, 50, 100, 500, 1000\}$. In our experiments, we normalize $dmm_i(k)$ to 1 by reporting

$$v = \max_{i \in \mathcal{T}} \{dmm_i(k)/k\} \tag{4.4}$$

That facilitates the comparison between system with different number of tasks and between $dmm_i(k)$ for different k.

4.2.2.1 Impact of k on the running time

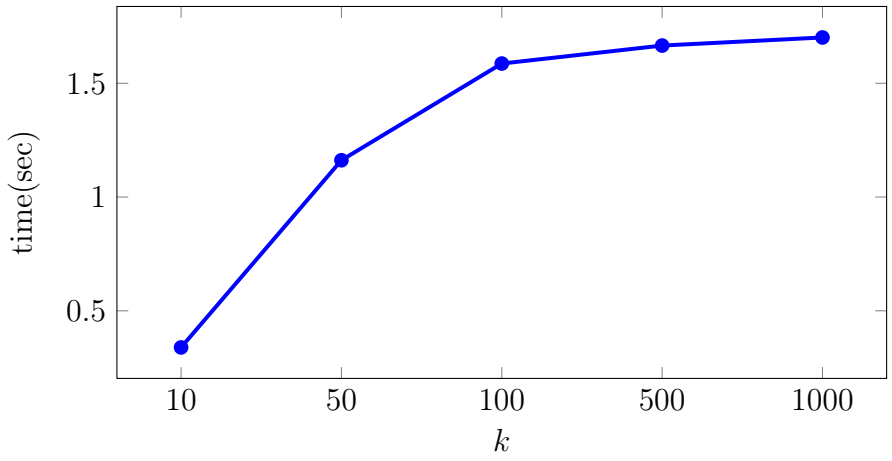

Figure 4.4: The average running time as a function of k.

Providing guarantees for large values of k is challenging for approaches that check all possible scenarios of a time window of k consecutive instances such as [113]. Figure 4.4 shows the average running time of computing $dmm_i(k)$ as a function of k. Each point on the curve is the average running time over the 1000 test cases with a given utilization $U \in \{0.4, 0.5, 0.6, 0.7, 0.8\}$.

The logarithmic shape of the curve is not surprising because $\Omega_k^{j \to i}$ increases logarithmically as k increases. As $\Omega_k^{j \to i}$ constrains the ILP in Equation 3.19, the size of the problem also grows logarithmically.

4.2.2.2 Impact of $\#\{\tilde{\mathcal{C}}\}$ on the running time

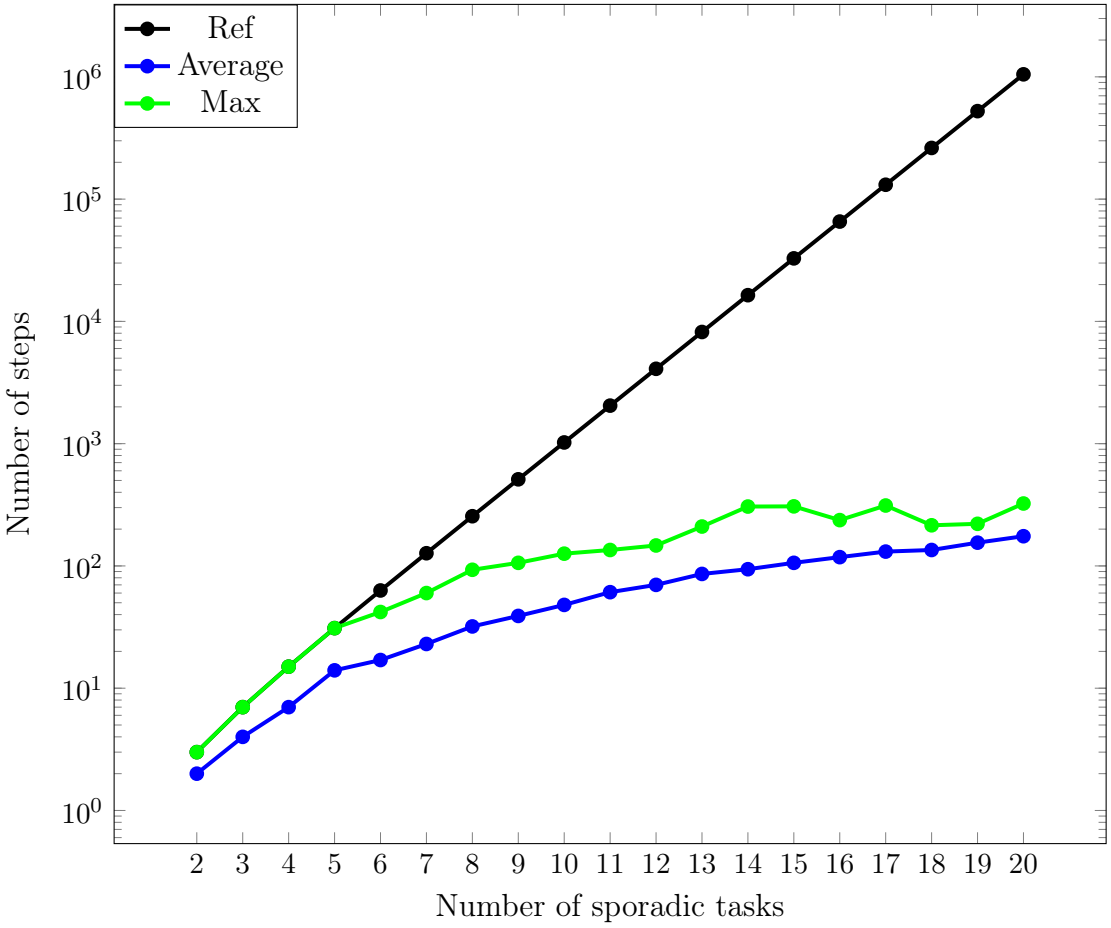

Figure 4.5: Number of steps that the LP solution needs to compute $dmm_i(k)$ w.r.t. the number of sporadic overload tasks.

The size of the set of unscheduable combinations $\{\tilde{\mathcal{C}}_i\}$ inflates exponentially

whenever the number of sporadic overload tasks increases. However, the LP relaxation presented in Algorithm 1 tackles this issue by relying on column generation to bound $dmm_i(k)$, which does not require to iterate over the set $\{\tilde{\mathcal{C}}_i\}$. This experiment underlines the efficiency of the LP relaxation by reporting the number of algorithm steps, see Algorithm 1, as a function of the number of sporadic tasks n_s.

Figure 4.5 presents the maximum and the median number of algorithm steps that were required to compute $dmm_i(k)$ in this experiment. The maximum possible number of steps for each value of n_s, i.e., 2^{n_s} because every step we add a new variable (unschedulable combination) to the LP, is plotted in black to illustrate the efficiency. Note that the y-axis is log-scaled. Even for $n_s = 20$, there is no need for more than 324 steps. To convince the reader more about the results, we plot the mode of algorithm steps, i.e., the value that appears most often, for each value of n_s.

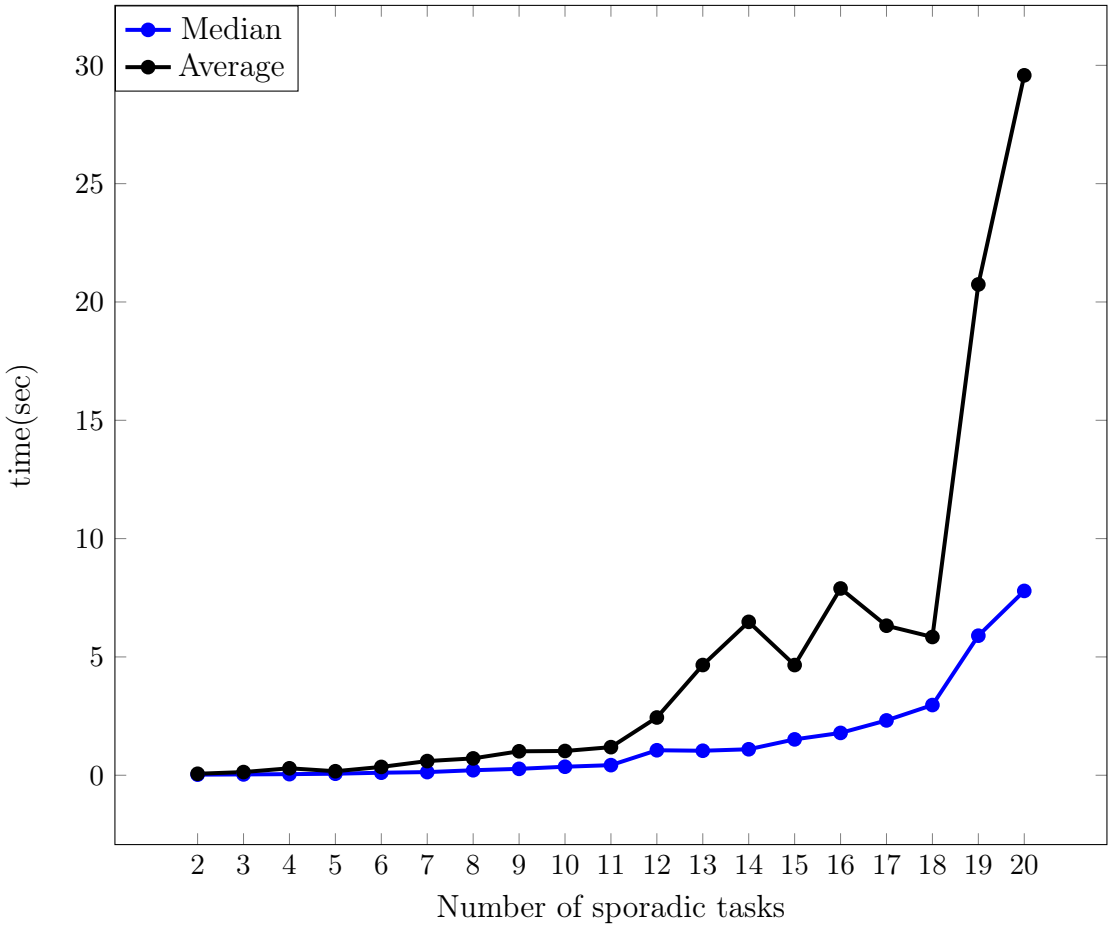

Figure 4.6: The average and the median of the running time as a function of n_s .

In addition to the number of steps in the LP relaxation algorithm, we also study (see Figure 4.6) the average and the median running time of computing $dmm_i(k)$ and the median as a function of n_s.

As expected, the curves grow exponentially. However, even for $n_s = 20$ the average does not exceed 29.578 seconds and 50% of test cases with $n_s = 20$ have a running time $\leqslant 7.789$ seconds.

Note that n the number of all tasks in a task set has no impact on the running time nor on the quality of results.

4.2.2.3 Impact of sporadic overload

In this thesis, we consider temporarily overloaded real-time systems in which $U \leqslant 1$ and at least one instance of a typical task misses its deadline. However, the considered sporadic overload may result in $\forall k \in \mathbb{N}^+ : dmm_i(k) = k$ due to the pessimism in TWCA. The sources of pessimism are presented in Section 4.1.2. In such a case, we say that the sporadic overload has a *permanent impact* on the computed $dmm(k)$ function. Otherwise, when $\exists k \in \mathbb{N}^+ : dmm_i(k) < k$, we say the sporadic overload has a *transient impact*. We show in this experiment how far TWCA can be useful to check the schedulability of WHRT tasks along with the share of $U_{\mathcal{O}}$. For this end, Table 4.1 shows the quartiles of v, which is the normalized value of $dmm_i(k)$ presented in Equation 4.4, for all the generated test cases but only for $k = 100$. The table is organized in 5 sub-tables; one per system utilization. Note that, $v = 1$ implies that the sporadic overload has a permanent impact on $dmm(k)$.

The table illustrates that the impact of the sporadic overload is relative to the total utilization U (one could also say that: it is relative to the typical utilization $U_{\mathcal{T}}$). For instance, when $U = 0.4$ and $U_{\mathcal{O}} = 0.06$ ($U_{\mathcal{O}}/U = 0.15$), the sporadic overload has a transient impact for 50% of the test cases because $v \leqslant 0.555$. However, when $U = 0.6$ and $U_{\mathcal{O}} = 0.06$ ($U_{\mathcal{O}}/U = 0.1$), the sporadic overload has a transient impact for only 25% of test cases with $v \leqslant 0.62$. For $U = 0.8$ any sporadic overload with $U_{\mathcal{O}} \geqslant 0.008$ may have a permanent impact.

The utilization of a sporadic overload task is defined as follows (see Definition 2.18):

$$U_j = \lim_{\Delta t \to \infty} \frac{\eta_j^+(\Delta t) \,.\, C_j}{\Delta t}$$

On the one hand, C_j has an influence on N_i and the schedulability of $\bar{c} : j \in \bar{c}$. On the other hand, $\eta_j^+(\Delta t)$ constrains the ILP through $\Omega_k^{j \to i}$. Hence, one can observe in general that the smaller the ratio $U_{\mathcal{O}}/U$ the fewer deadlines will be missed. However, for a given U_j the deadline miss model of τ_i may change depending on the specifications of C_j and $\eta_i^+(\Delta t)$. A long execution time of τ_j may increase N_i,

U	0.4					
$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.01	0.01	0.02	0.03	0.02	0.05
Q1	0.02	0.04	0.1	0.19	0.21	0.3525
Q2	0.0324	0.08	0.19	0.38	0.555	0.8935
Q3	0.07	0.16	0.4537	0.84	1	1
Max	0.9071	1	1	1	1	1
U	0.5					
$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.01	0.01	0.01	0.05	0.05	0.02
Q1	0.02	0.07	0.18	0.2675	0.38	0.61
Q2	0.035	0.145	0.425	0.69	0.895	1
Q3	0.1	0.39	0.9925	1	1	1
Max	1	1	1	1	1	1
U	0.6					
$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.01	0.02	0.04	0.05	0.04	0.1
Q1	0.03	0.09	0.26	0.62	0.9	0.915
Q2	0.0708	0.2312	0.76	1	1	1
Q3	0.1412	0.5187	1	1	1	1
Max	0.64	1	1	1	1	1
U	0.7					
$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.01	0.02	0.07	0.03	0.1	0.02
Q1	0.04	0.1725	0.63	1	1	1
Q2	0.11	0.4482	1	1	1	1
Q3	0.2525	1	1	1	1	1
Max	1	1	1	1	1	1
U	0.8					
$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.01	0.01	0.06	0.14	0.05	0.04
Q1	0.1	0.4	1	1	1	1
Q2	0.2	1	1	1	1	1
Q3	0.5975	1	1	1	1	1
Max	1	1	1	1	1	1

Table 4.1: Sporadic overload impact. Quartiles of v for $k = 100$ under FPP scheduling policy.

but at the same time there will be fewer instances of τ_j which implies that fewer busy-windows will be impacted. In contrast, a short execution time may decrease N_i but in the favor of increasing $\eta_i^+(\Delta t)$. The problem of budgeting for sporadic overload tasks under the constraint that periodic tasks must satisfy a given DMM is the topic of Chapter 6.

Despite the pessimism introduced by our technique for computing $dmm_i(k)$, TWCA can validate many more systems than the standard worst-case analysis. For instance, if tasks can tolerate no more than $m = 6$ out of $k = 100$, TWCA can accept 530 test cases in these experiments, which is about 10.6% while the worst-case analysis rejects all of them as unschedulable.

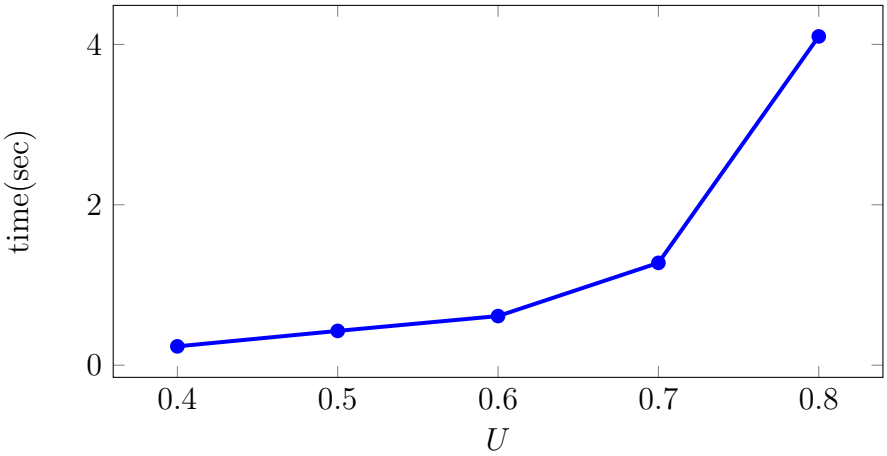

Figure 4.7: The average running time as a function of the utilization.

We now study the running time of the proposed analysis. Figure 4.7 shows the average running time of the computation of $dmm_i(k)$ as a function of U. Each point on the curve is the average running time over the 1000 test cases with a given utilization for $k \in \{10, 50, 100, 500, 1000\}$.

As expected, the running time increases exponentially with U due to the increasing number of unschedulable combinations (when U is large, any sporadic overload causes deadline-misses). However, even for $U = 0.8$ the average running time does not exceed 4 seconds so TWCA can still be applied for systems of that size.

4.2.2.4 Impact of priority assignment (Rate Monotonic)

Priority assignment is known to play a crucial role in ensuring schedulability. It does therefore also impact the quality of computed DMMs. This experiment considers the Rate Monotonic (RM) approach to assign priorities. In RM: the

$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.01	0.01	0.02	0.02	0.03	0.01
Q1	0.02	0.0425	0.1	0.1681	0.2375	0.26
Q2	0.04	0.11	0.2391	0.4097	0.5175	0.6437
Q3	0.1192	0.216	0.42	0.8927	1	1
Max	1	1	1	1	1	1

Table 4.2: Impact of priority assignment. Quartiles of v for $U = 0.4$ and $k = 100$ under RM scheduling policy.

shorter the period, the higher the priority. Sporadic overload tasks have the highest priorities in the task set to interfere with all tasks in $\mathcal{T}$.

The experiments reported here are for 1000 test cases generated as explained in Section 4.2.1 for $U = 0.4$. We choose a low utilization because we are not willing to study the impact of U in this experiment. Therefore, we reduce this impact by setting $U = 0.4$. Table 4.2 shows the quartiles of the reported value $v = \max_{i \in \mathcal{T}}\{dmm_i(k)/k\}$. v is slightly improved over the random assignment in Table 4.1 for $U_{\mathcal{O}}/U = 0.15$ and 0.2.

RM does not improve significantly over random assignment when a deadline-miss occurs. The values of v in Table 4.2 are sometimes even larger than those in 4.1 for $U = 0.4$. This is surprising because RM is proven to be an optimal priority assignments for FPP. We want to study this further, and for this end we study the number of preemptions. It is known that RM increases the number of preemptions [17] which implies that τ_i gets delayed more and more (and thus N_i increases) as the number of preemptions increases. Thus, the key point here is N_i. Table 4.5 shows the quartiles of N_i for FPP with random priority assignment and RM with $U = 0.4$. One can observe that RM yields more deadline-misses within one busy-window than a random priority assignment. Remember that N_i is a source of pessimism in computing $dmm_i(k)$, see Section 4.1.2.

4.2.2.5 Priority assignment impact (Deadline Monotonic)

This experiment considers the Deadline Monotonic (DM) approach to assign priorities. In DM: the shorter the relative deadline, the higher the priority. Sporadic overload tasks have the highest priorities in the task set.

Another 1000 synthetic test cases are generated as explained in Section 4.2.1 for $U = 0.4$. Table 4.3 shows the quartiles of $v = \max_{i \in \mathcal{T}}\{dmm_i(k)/k\}$. There is no improvement over the random assignment of priorities, see Table 4.1.

Table 4.5 shows the quartiles of N_i for DM with $U = 0.4$. Just like for RM, DM yields more deadline-misses within one busy-window than a random priority

$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.01	0.01	0.02	0.02	0.03	0.03
Q1	0.02	0.04	0.12	0.16	0.245	0.23
Q2	0.0438	0.1	0.28	0.45	0.6	0.66
Q3	0.1052	0.2441	0.5743	0.96	1	1
Max	1	1	1	1	1	1

Table 4.3: Impact of priority assignment. Quartiles of v for $U = 0.4$ and $k = 100$ under DM scheduling policy.

assignment.

4.2.2.6 Preemption impact

$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.02	0.02	0.04	0.04	0.06	0.04
Q1	0.04	0.06	0.12	0.2175	0.28	0.3292
Q2	0.06	0.1	0.22	0.39	0.4763	0.5887
Q3	0.1126	0.18	0.38	0.651	0.86	1
Max	0.63	1	1	1	1	1

Table 4.4: Preemption impact. Quartiles of v for $U = 0.4$ and $k = 100$ under FPNP scheduling policy.

The previous two experiments underline the impact of preemptions on the quality of the computed DMMs. We continue this discussion further by studying non-preemptive fixed-priority scheduling. We studied FPNP and presented an analysis to compute a DMM under FPNP in Section 3.2.

The experiments presented in this section are based on 1000 test cases generated as explained in Section 4.2.1 with a total utilization $U = 0.4$. Table 4.4 shows the quartiles of the reported value $v = \max_{i \in \mathcal{T}}\{dmm_i(k)/k\}$. Table 4.2, when compared with Table 4.1, shows that the FPNP policy yields better results than FPP (with RM, DM or random priority assignment). Table 4.5 provides an explanation for this: For 75% of the test cases, $N_i = 1$ in the FPNP case, which eliminates a key source of pessimism in the DMM computation.

At this point, we can propose the following conclusions about the efficiency of our analysis for fixed priority scheduling.

- TWCA reports pessimistic bounds.
- TWCA scales well w.r.t. k and n_s.

	FPNP	FPP	RM	DM
Min	1	1	1	1
Q1	1	1	1	1
Q2	1	1	2	2
Q3	1	2	3	4
Max	9	18	41	30

Table 4.5: Quartiles of N_i for $U = 0.4$

- The LP relaxation in Algorithm 1 is sufficiently efficient.
- TWCA implies better to nonpreemptive scheduling.
- TWCA implies better to a relatively low transient sporadic overload.
- TWCA can be used to check whether a sporadic overload has a transient or a permanent impact.

4.2.3 EDF scheduling policy

In this section, we study how TWCA performs for systems scheduled with EDF.

We generated 5000 synthetic test cases depending on the general setting up in Section 4.2.1. The total utilization covers $U = 0.9$ besides the range of values mentioned in the general setting up. Each sporadic overload task has a relative deadline equal to its worst-case execution time to enforce the typical tasks to miss their deadlines because the cases in which the task set is schedulable are not interesting for our experiments.

DMM was computed for each typical task $\tau_i \in \mathcal{T}$ for

$$k = \eta^+(\Delta) : \Delta = \{1, 10, 50, 100\} \times |BW^+|,$$

where $|BW^+|$ is the length of the longest busy-window. Zhang and Burns showed in [126] that a task may have hundreds of instances in the busy-window. That helps to do a fair comparison between tasks. Then for each test case, the value $v = \max_{i \in \mathcal{T}}\{dmm_i(k)/k\}$ was reported.

For the schedulability test, we implemented the Quick convergence Processor-demand Analysis (QPA) algorithm [126] that reduces significantly the number of iterations required to check the schedulability of a given task set. The QPA algorithm is presented in Theorem 5 in [126], and it applies for periodic or sporadic tasks (defined by the minimum inter-arrival time) with arbitrary deadline. Thus, we need first to prove that it applies to tasks with arbitrary arrival curves. Theorem 5 presented for a given task set $\mathcal{Z}$ the following algorithm:

$t \leftarrow \max\{d_i \mid d_i < L\}$;
while $(h(t) \leqslant t \wedge h(t) > d_{min})$
$\{$*if* $(h(t) < t)$ $t \leftarrow h(t)$;
else $t \leftarrow \max\{d_i \mid d_i < t\};\}$
if $(h(t) \leqslant d_{min})$ the task set is schedulable;
else the task set is not schedulable;
where:
L is the length of the worst-case busy-window, i.e., $L = |BW^+|$.
$d_{min} = \min_{i \in \mathcal{Z}}\{D_i\}$.
$h(t) = \sum_{i \in \mathcal{Z}} \max\{0, 1 + \lfloor \frac{t - D_i}{T_i} \rfloor\}$ is the demand bound function of periodic tasks where T_i denotes the period of τ_i.

To prove that the above algorithm applies to arbitrary arrival curves we only need to prove that the demand bound function applies to arbitrary arrival curves. Fortunately, this has been proved in, e.g., [43]. Therefore, we rewrite the above algorithm using the function $dbf(t)$, which is given in Equation 3.70, for our tasks that are modelled using arbitrary arrival curves. The algorithm is illustrated in Figure 4.8, where $d^{\Delta} = \max\{d_i \mid 0 < d_i < |BW^+| \wedge dbf(d_i) > d_i\}$.

Algorithm 3: QPA algorithm for tasks that are modelled using arbitrary arrival curves, based on [126].

```
1  t ← max{d_i | d_i < |BW+|};
2  while dbf(t) ⩽ t ∧ dbf(t) > d_min do
3      if dbf(t) < t then
4          t ← dbf(t);
5      else
6          t ← max{d_i | d_i < t};
7  if dbf(t) ⩽ d_min then
8      the task set is schedulable;
9  else
10     the task set is not schedulable;
```

4.2.3.1 Sporadic overload impact

Similarly to the experiment in Section 4.2.2.3, the quartiles of v are presented in Table 4.6 for each value of the sporadic overload share $U_{\mathcal{O}}/U$. EDF provides quite an improvement on DMMs. The reason is that, once a tasks misses its deadline,

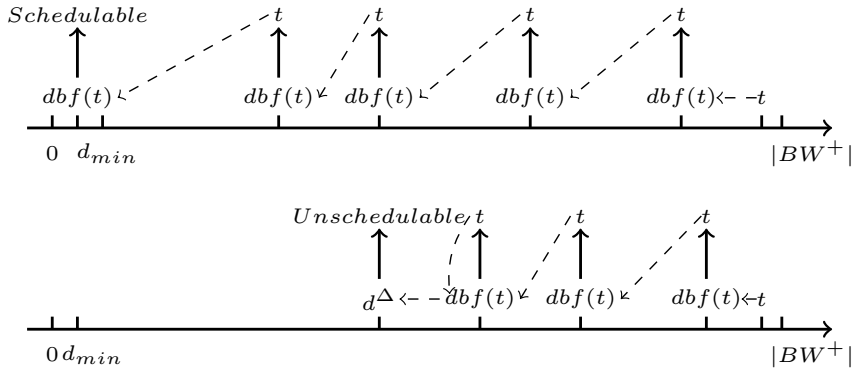

Figure 4.8: QPA algorithm, based on [126].

it gets a high priority, which helps keep N_i small and thus $dmm_i(k)$ is improved. Another reason is that EDF can schedule a load up to $U = 1$, which impacts the schedulability of sporadic overload combinations (a lot of combinations will be simply schedulable). TWCA, therefore, applies perfectly to EDF. Nevertheless, when a sporadic overload has a permanent impact $v = 1$ other approaches have to be used, e.g., RED [19] and D^{over} [61], to handle such overload conditions.

EDF scheduling sacrifices no task, instead, it fairly distributes the deadline-misses. Therefore, it is not possible to predict which task will miss its deadline. The real-time architect needs an analysis like TWCA not only to compute the DMMs but also to predict the tasks that may miss their deadline.

4.2.3.2 Domino effect

Under overload conditions it is possible that all tasks miss their deadlines, this phenomenon is called domino effect. When the system utilization $U > 1$, the domino effect may last forever. In this work we are interested in systems for which $U \leqslant 1$ as discussed in Chapter 2. The domino effect represents an undesirable behavior that must be avoided. In order to avoid domino effects, the operating system and the scheduling algorithm must be explicitly designed to handle domino effects in a controlled fashion [18]. Algorithms like RED [19] and D^{over} [61] are designed to detect overload conditions and eliminate domino effects by rejecting/postponing few instances in order to minimize the damage due to a deadline-miss.

The domino effect could be defined as follows: *If* $\forall i \in \mathcal{Z} : N_i > 0$*, we say the system may suffer from the domino effect.*

The above necessary condition can be used to conservatively predict the occurrence of the domino effect. The dynamic and unpredictable behavior of embedded real-time systems will eliminate an off-line sufficient and necessary condition for

U	0.4					
$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.004	0.0067	0.0133	0.015	0.02	0.016
Q1	0.007	0.016	0.04	0.05	0.0725	0.0546
Q2	0.0133	0.024	0.0533	0.0883	0.12	0.0955
Q3	0.0193	0.032	0.08	0.1308	0.2	0.1588
Max	0.06	0.14	0.22	0.3067	0.7	0.53
U	0.5					
$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.0025	0.005	0.01	0.0133	0.0133	0.012
Q1	0.008	0.015	0.0437	0.06	0.1	0.1
Q2	0.012	0.025	0.07	0.095	0.1925	0.19
Q3	0.02	0.04	0.12	0.18	0.3383	0.4167
Max	0.06	0.1	0.48	0.49	1	1
U	0.6					
$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.0025	0.0033	0.01	0.0067	0.005	0.02
Q1	0.0067	0.0167	0.05	0.08	0.1465	0.1525
Q2	0.012	0.03	0.0856	0.15	0.24	0.2854
Q3	0.02	0.05	0.15	0.32	0.5007	0.6667
Max	0.08	0.2933	0.56	0.9	1	1
U	0.7					
$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.0026	0.0055	0.0171	0.0183	0.01	0.0267
Q1	0.0067	0.02	0.075	0.1227	0.215	0.2519
Q2	0.0143	0.04	0.1515	0.2825	0.636	0.5715
Q3	0.0283	0.0735	0.278	0.555	0.9757	1
Max	0.06	0.2	0.8	1	1	1
U	0.8					
$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.0021	0.0011	0.0093	0.0133	0.0078	0.0076
Q1	0.0075	0.04	0.135	0.2444	0.3012	0.474
Q2	0.02	0.0727	0.2571	0.5333	0.686	1
Q3	0.03	0.112	0.48	1	1	1
Max	0.14	1	1	1	1	1
U	0.9					
$U_{\mathcal{O}}/U$	0.001	0.01	0.05	0.1	0.15	0.2
Min	0.001	0.0025	0.0031	0.0476	0.0313	0.0571
Q1	0.02	0.06	0.2724	0.505	0.6767	1
Q2	0.032	0.1141	0.5699	1	1	1
Q3	0.0514	0.23	1	1	1	1
Max	0.18	1	1	1	1	1

Table 4.6: Sporadic overload impact. Quartiles of v under EDF scheduling policy, based on [51].

Message	P	J	C	θ	D	R^+
τ_1	40	2	6	2	38	26
τ_2	40	2	6	3	38	20
τ_3	40	20	4	4	20	12
τ_4	100	20	6	3	80	20

Table 4.7: Timing characteristics of considered tasks under WRR scheduling policy. P,J, C, θ, and D denote respectively: period, jitter, worst-case execution time, time slot, relative deadline. Based on [48].

predicting the domino effect. An alternative solution would be to analyze all possible busy-windows within which all tasks participate. If in one busy-window all tasks miss the deadline, then the domino effect occurs. Although it is a tighter condition, this solution does not seem feasible because the size of such set will be extremely huge. In this work, thus, we adopt the necessary condition in the above lemma to safely predict the occurrence of the domino effect.

Our experiments, based on the above necessary condition, show that about 40% of the generated task sets will never experience a domino effect. In other words, we can say that in the worst-case the guarantees computed by TWCA are sufficient to describe the behavior with the presence of transient overload for 40% of systems that follow our system model. Keep in mind that we conservatively predict the occurrence of the domino effect, which makes these results pessimistic and therefore TWCA should be in practice more useful.

4.2.4 WRR scheduling policy

In this experiment the general setting up was relaxed such that a single resource with 4 tasks scheduled according to WRR policy is considered with fixed timing characteristics. This task set is derived from the industrial challenge presented in WATERS 2015 by TRT [54]. The timing characteristics are shown in Table 4.7. The worst-case response time analysis shows that no task misses its deadline, see the most right column in Table 4.7.

The given model represents the typical model of the system and we want to study how sporadic overload would impact its schedulability, thus, we added 4 synthetic sporadic overload tasks as follows: Each overload task τ_s has $C_s = 2, \theta_s = 2$. The typical utilization is $U_{typ} \approx 0.59$, the overload utilization was budgeted as follows:

$$U_{over} = (a/(1-a)) \ * \ U_{typ}.$$

where $a = U_{\mathcal{O}}/U$ and $U_{\mathcal{O}}/U \in \{0.001, 0.01, 0.05, 0.1, 0.15, 0.2\}$ as in the general

setting up.

The experiment was repeated 5000 times. In each test case a new synthetic sporadic overload was generated. Every set of 1000 test cases has the same share $U_{\mathcal{O}}/U$. $dmm_i(k)$ was computed for the 4 tasks in the system with $k = 100$.

4.2.4.1 Sporadic overload impact

U_{over}/U_{typ}	0.001				0.01			
Message	τ_1	τ_2	τ_3	τ_4	τ_1	τ_2	τ_3	τ_4
Min	0	0	0.02	0	0.04	0.04	0.04	0
Q1	0	0	0.04	0	0.1	0.1	0.14	0
Q2	0.02	0.02	0.05	0	0.12	0.12	0.21	0
Q3	0.03	0.03	0.05	0	0.1	0.13	0.25	0
Max	0.06	0.06	0.12	0	0.24	0.27	0.33	0
U_{over}/U_{typ}	0.05				0.1			
Message	τ_1	τ_2	τ_3	τ_4	τ_1	τ_2	τ_3	τ_4
Min	0.06	0.1	0.12	0	0.08	0.08	0.34	0
Q1	0.2675	0.25	0.53	0	0.57	0.38	0.91	0
Q2	0.36	0.3	0.63	0	0.83	0.56	1	0
Q3	0.45	0.37	0.74	0	1	0.93	1	0
Max	1	0.9	1	0	1	1	1	0
U_{over}/U_{typ}	0.15				0.2			
Message	τ_1	τ_2	τ_3	τ_4	τ_1	τ_2	τ_3	τ_4
Min	0.12	0.12	0.32	0	0.21	0.12	0.26	0
Q1	1	0.69	1	0	1	0.99	1	0
Q2	1	1	1	0	1	1	1	0
Q3	1	1	1	0	1	1	1	0
Max	1	1	1	0	1	1	1	0

Table 4.8: Sporadic overload impact. Quartiles of $dmm_i(100)/100$ under WRR schededuling policy.

Table 4.8 shows the quartiles of $dmm_i(k)/k$ for the four tasks. While τ_4 meets all its deadline, almost every sporadic overload with a share of $U_{\mathcal{O}}/U \geqslant 0.1$ have a permanent impact ($v = 1$) on τ_3, i.e., al instances of τ_3 miss their deadline.

To explain the results in Table 4.8 let us look at Figure 4.9 that shows the average of $dmm_i(100)$ over 1000 test cases for each value of U_{over}/U. Figure 4.9 shows that τ_4 misses no deadline. τ_1 misses more deadlines than τ_2, and τ_3 misses more deadlines than τ_1 and τ_2. To explain that, we need to look at Table

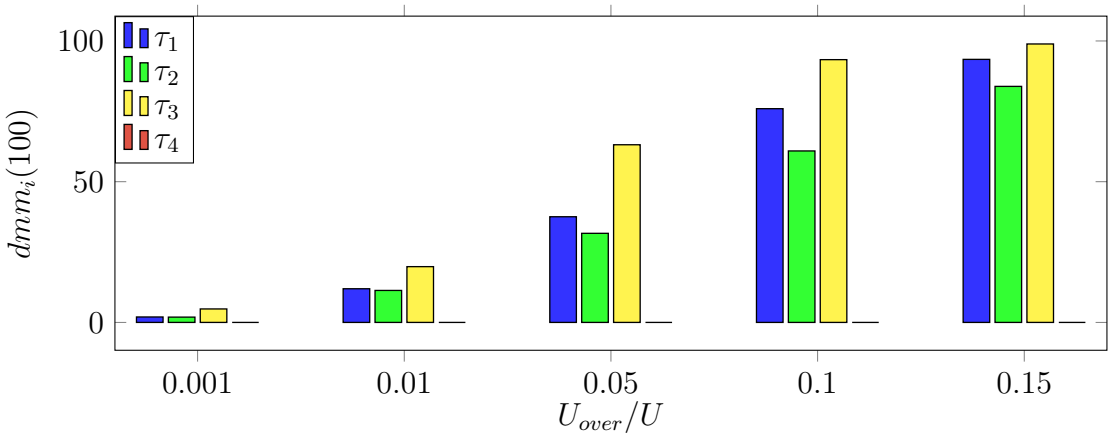

Figure 4.9: The scheduling policy (WRR) impact on DMMs.

4.7 where the one observe that τ_1 has $C_1/\theta_1 = 3$, τ_2 has $C_2/\theta_2 = 2$ and τ_3 has $C_3/\theta_3 = 1$, where C_i is the worst-case execution time of τ_i and θ_i is the time slot. This ratio implies that each instance of τ_i may get at most an interference of (C_i/θ_i) . $(WRR_{turn} - \theta_i)$, in other words, a message with large C_i/θ_i will suffer from any overload instance more than messages with small C_i/θ_i. However, τ_3 has relatively shorter deadline than τ_1 and τ_2, which implies that it has larger set of unschedulable combinations $\tilde{\mathcal{C}}_3$.

4.2.5 A comparison between the impact of sporadic overload on different scheduling policies

From the experiments that studied the impact of sporadic overload, one can observe that the impact varies depending on the scheduling policy. To emphasize this point with a fair comparison between the considered scheduling policies, we design a new experiment. This experiment recalls the second quartile Q2 from Tables 4.1, 4.6 and 4.8 for FPP (arbitrary priority assignments), EDF and WRR (τ_3) for $U = 0.6$. 1000 synthetic test cases were generated under FPNP scheduling policy with $U = 0.6$ and the same for RM and DM. Figure 4.10 presents Q2 as a function of U_{over}/U for the considered scheduling policies.

While EDF has the best curve, i.e., tasks are slightly impacted by the sporadic overload and they therefore have low number of deadline-misses comparing to other scheduling policies, FPP has the worst. However, EDF suffers from the unpredictability and the domino effect. Sporadic overload has less impact on tasks under FPNP comparing to FPP, RM, and DM because it (FPNP) causes smaller N_i as we showed in Table 4.5. FPP (arbitrary priority assignments), RM and DM have a high level of isolation between tasks such that hard real-time tasks can

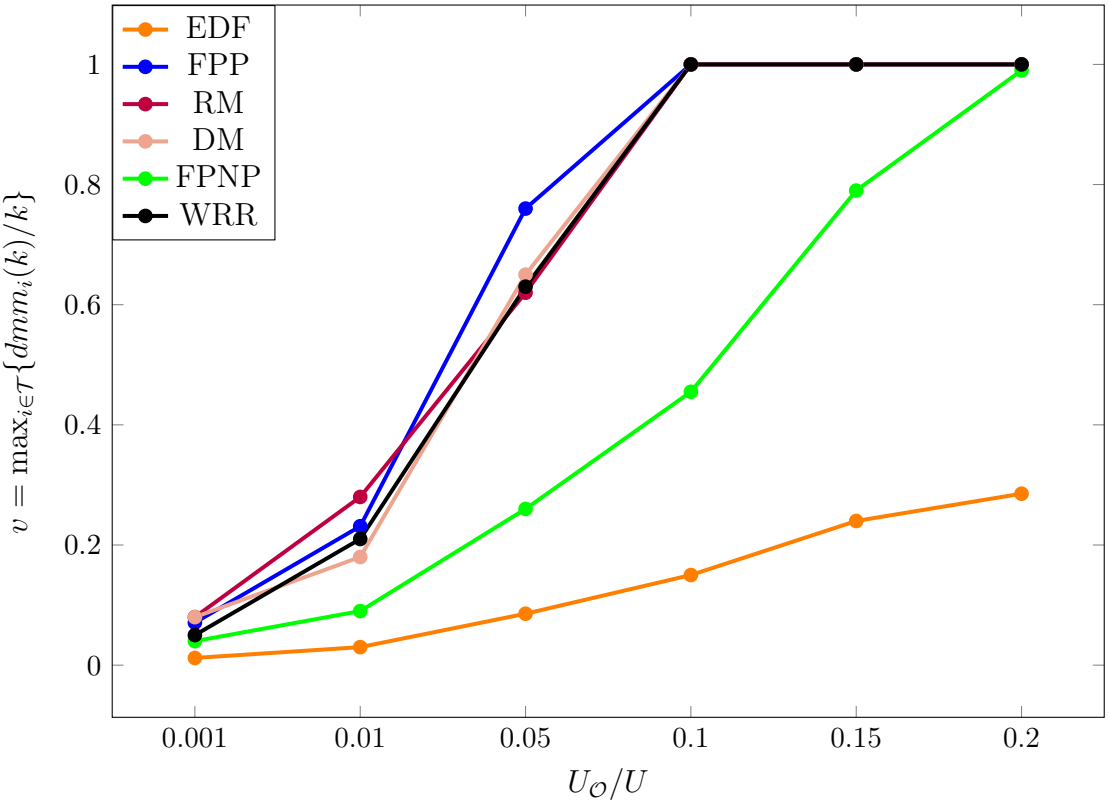

Figure 4.10: How the scheduling policy alters the sporadic overload impact.

meet their deadline. Remember that the results reported in Figure 4.10 represent the maximum number of deadline-misses $v = \max_{i \in \mathcal{T}}\{dmm_i(k)/k\}$.

This experiment can be useful in the design process of WHRT system to assist the system architect in selecting a proper scheduling policy.

4.3 Summary

In this chapter, we underlined some properties of $dmm_i(k)$, then we analytically evaluated the complexity of computing a DMM using the proposed analysis. We addressed the sources of pessimism in computing DMMs as well, and we showed the impact of the schedulability criterion on the quality of computed DMMs.

Throughout the the experiments, we studied the efficiency, scalability and applicability of TWCA. A synthetic test case generator was developed in order to achieve the evaluation; we generated a total of 20000 synthetic test cases. The experiments covered the FPP, RM, DM, FPNP, WRR and EDF scheduling poli-

cies. We investigated: 1) How large the running time of our analysis is, and how it evolves as a function of k or of the number of unschedulable combinations. 2) How the amount of sporadic overload impacts the provided guarantees.

The experiments show that TWCA reports pessimistic bounds. However, it scales well w.r.t. k and n_s. The experiments demonstrate that the impact of sporadic overload varies depending on the scheduling policy. In general, TWCA applies better to a relatively low transient sporadic overload.

We present more experiments in Chapter 5 and Chapter 6 to evaluate the two extensions.

5 DEADLINE MISS MODELS FOR TASK CHAINS

Real-time systems with functional dependencies between tasks (that is, such that task can activate each other) often require end-to-end (as opposed to task-level) guarantees. Many such systems are weakly-hard in the sense that they can accept longer end-to-end delays if one can bound their frequency.

The analysis presented in Chapter 3 and Chapter 4, does not cover task dependencies. This represents a limitation to its applicability in practice, as motivated by a case study directly derived from industrial practice at Thales Research & Technology (TRT). In this chapter, we present an extension of TWCA to provide end-to-end DMMs. To achieve that we exploit task chain properties that arise from the priority assignment of tasks in FPP systems. This extension has been published in [49].

This chapter is organized as follows: first, we present a case study provided by TRT as a motivation for this extension, and we formulate the addressed problem. Section 5.2 revisits the worst-case latency analysis proposed by [105], which represents a foundation for our analysis. The extension of TWCA will be presented in Section 5.3. Section 5.4 shows our experimental results.

5.1 Case Study and Problem Statement

This chapter is motivated by and validated on an industrial case study provided by TRT. Figure 5.1 represents the system to be analyzed where tasks have precedence constraints forming task chains. The system shown in Figure 5.1 is temporarily overloaded by two sporadic task chains σ_a and σ_b and WHRT guarantees are needed for chains σ_c and σ_d. Therefore, we present a method to compute *end-to-end DMMs* for FPP systems with task chains. This bounds the number of potential deadline-misses in a given sequence of executions of a task chain. The approach is an extension of TWCA, for which we exploit task chain properties derived from the priority assignment of tasks in a way similar to [105].

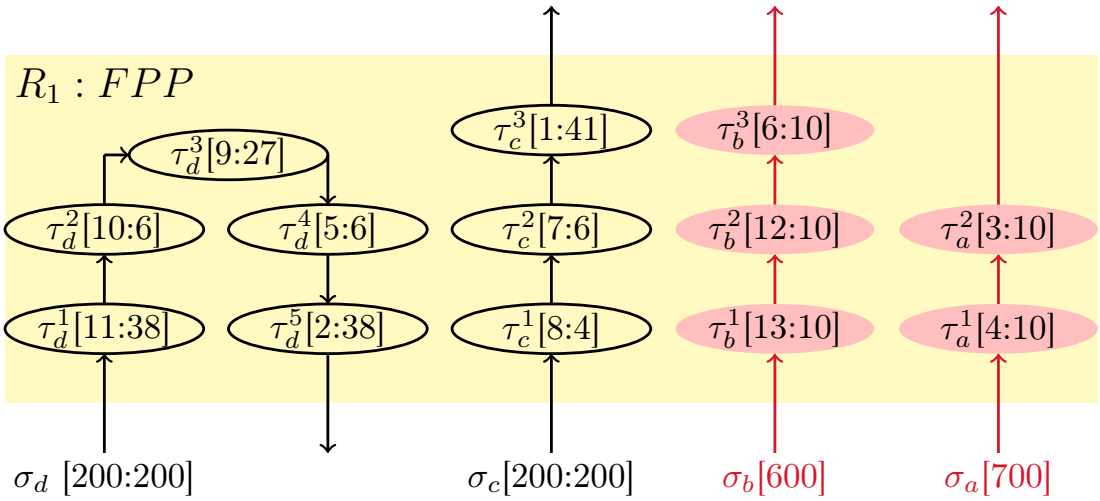

Figure 5.1: Model of the case study. The following notations are used: task chains are specified as $\sigma[\delta^-(2) : D]$ and tasks with $\tau[\pi : C]$. Chains σ_c and σ_d are periodically activated while σ_a and σ_b are sporadic, overload chains.

5.1.1 System model

In this section, we enrich the system model presented in Chapter 2 with task dependencies. We consider uniprocessor systems consisting of a finite set of m disjoint *task chains* scheduled according to the FPP scheduling policy. A task chain is a sequence of distinct tasks, which activate each other. Tasks in a system are required to belong to exactly one chain. Note that, neither fork/join nor cycles are in the scope of this work.

Definition 5.1. *A task chain σ_a, $a \in [1, m]$, is defined by a finite sequence $(\tau_a^1, \tau_a^2, \ldots, \tau_a^{n_a})$ of distinct tasks for some $n_a \in \mathbb{N}^+$, meaning that the output of τ_a^i is connected to the input of τ_a^{i+1} for $i \in [1, n_a - 1]$. Every task chain σ_a is assigned*

- *an* activation model *defining the frequency of arrival at the input of τ_a^1;*
- *an arbitrary* relative deadline D_a.

The tasks in σ_a are denoted τ_a^i, τ_a^j etc. Task τ_a^i denotes the i-th task in task chain σ_a. The number of tasks in σ_a is denoted n_a. The first task in σ_a, i.e. τ_a^1, is called the *header task* of σ_a and the last one, i.e. $\tau_a^{n_a}$, is called the *tail task*.

Figure 5.2 shows an example system with two task chains:

$$\sigma_a = (\tau_a^1, \tau_a^2, \tau_a^3, \tau_a^4, \tau_a^5, \tau_a^6), \quad \sigma_b = (\tau_b^1, \tau_b^2, \tau_b^3).$$

We denote $\mathcal{C}$ the set of task chains. This set is partitioned into $\mathcal{SC}$ and $\mathcal{AC}$, which contain respectively the *synchronous* and *asynchronous* chains. Synchronous and asynchronous chains are specified in the same way but behave differently at execution: In a synchronous chain σ_a an incoming instance cannot be processed until the previous instances of σ_a have finished [105]. In an asynchronous chain σ_b an incoming instance is processed independently from previous instances.

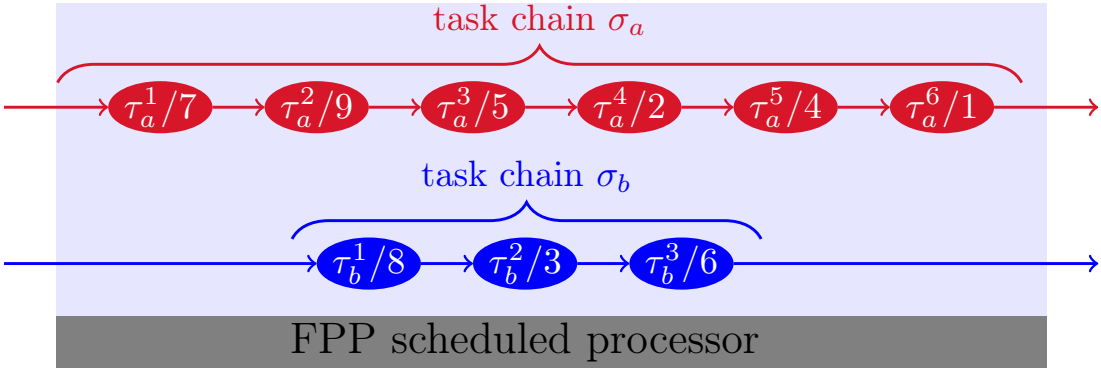

Figure 5.2: A system task structure with chains and task priorities. $\tau_a^5/4$ denotes the fifth task in the task chain σ_a with a priority equal to 4.

The activation models of task chains are defined using *arrival curves* i.e., functions $\eta_a^-(\Delta T), \eta_a^+(\Delta T)$, and the pseudo-inverse representation of arrival curves $\delta_b^-(k), \delta_b^+(k)$, see Definition 2.7.
A task τ_a^i is defined by: (1) an arbitrary priority π_a^i and (2) an upper bound on its execution time C_a^i (we take 0 as a lower bound).
The timing behavior of a task τ_a^i is as described in Chapter 2. Preemption delays are due to the task being blocked by higher priority tasks from other task chains, but also by higher priority tasks from the same chain if it is asynchronous. In contrast, tasks in a synchronous chain cannot be preempted by other tasks of the same chain, even if they have higher priority. Task τ_a^i finishes latest after having been scheduled for C_a^i units of time.

The timing behavior of a task chain σ_a is an infinite sequence of task chain instances, where a task chain instance is made of one instance of each task in the chain such that the finish time of task τ_a^i corresponds to the activation time of task τ_a^{i+1} (assuming τ_a^i is not the last task in the chain). The activation of task τ_a^1 follows the activation model of σ_a.

The *latency* of an instance of a task chain σ_a is the time interval between the activation of the header task of σ_a and the finish time of the tail task of σ_a. The *worst-case latency* of σ_a is the longest latency over all instances of σ_a. An instance of σ_b is said to *miss its deadline* if its latency exceeds the relative deadline of

σ_b. Similarly to what is considered in Chapter 3, we consider a deadline-agnostic scheduler.

As usual in TWCA, we suppose that deadline-misses are caused by rarely activated sporadic chains, e.g., interrupt service routines or recovery chains. These chains cause transient overload, increasing chain latencies which may cause deadline-misses, hence their name: *overload chains*. We assume that the set of overload chains is identified and denoted $\mathcal{C}_{over}$.

Definition 5.2. *A* deadline miss model *for a task chain σ_b is a function $dmm_b : \mathbb{N}^+ \to \mathbb{N}$ such that $dmm_b(k)$ bounds the maximum number of deadline-misses in a window of k consecutive executions of σ_b.*

In this chapter, we address the problem of computing DMMs of task chains in temporarily overloaded systems.

5.2 Latency Analysis Revisited

Let us first revisit the worst-case latency analysis of systems with task chains presented in [105]. Consider two chains σ_a and σ_b. To quantify the interference of σ_a on σ_b we distinguish two cases:

1. *some* tasks in σ_a have lower priority than *all* tasks in σ_b; in that case, σ_a will be blocked by σ_b every time it reaches one of those tasks.

2. In any other case, σ_a is said to *arbitrarily interfere with* σ_b. This means that every time σ_a is triggered, we suppose that it may entirely execute before σ_b can be scheduled again. As we will see later, there is no guarantee however that this will happen.

Definition 5.3. *A chain σ_a is said to be* deferred by *chain σ_b if*

$$\exists i \in [1, n_a], \pi_a^i < \min_{1 \leqslant j \leqslant n_b} \{\pi_b^j\}$$

Otherwise it is arbitrarily interfering with σ_b.

The set of chains deferred by σ_b is denoted $\mathcal{DC}(b)$ and the set of chains arbitrarily interfering with σ_b is denoted $\mathcal{IC}(b)$.

For a chain σ_a that is arbitrarily interfering with σ_b, interference on σ_b can be directly derived from the number of instances of σ_a. If σ_a is, however, deferred by σ_b, then interference is defined based on the concept of *segment* of σ_a w.r.t. σ_b. Intuitively, a segment of σ_a w.r.t. σ_b represents a subchain of σ_a that may interfere with σ_b.

Definition 5.4. *A* segment of σ_a w.r.t σ_b *is a maximal subchain* $(\tau_a^i, \tau_a^{i+1}, \ldots, \tau_a^{i+k})$ *of* σ_a*,* $i \in [1, n_a]$ *and* $k \in [0, n_a{-}1]$*, with the convention*[1] *that task identifiers should be read modulo* n_a *and such that*

$$\forall l \in [0, k], \pi_a^{i+l} \geqslant \min_{1 \leqslant j \leqslant n_b} \{\pi_b^j\}$$

Note that we (conservatively) assume that a segment may span over two instances of σ_b. S_a^b denotes all such segments.

Example 5.1. *Chain* σ_a *in Figure 5.2 has 2 segments w.r.t. chain* σ_b*:* $(\tau_a^1, \tau_a^2, \tau_a^3)$ *and* (τ_a^5)*. Note that* τ_a^4 *and* τ_a^6 *have lower priority than* τ_b^2 *and are therefore not part of any segment.*

Definition 5.5. *The* critical segment *of a chain* σ_a *deferred by* σ_b*, denoted* $s_{a,b}$*, is the segment* $(\tau_a^i, \tau_a^{i+1}, \ldots, \tau_a^{i+k})$ *of* σ_a *w.r.t.* σ_b *that maximizes computation time, i.e.,* $\sum_{0 \leqslant l \leqslant k} C_a^{i+l}$*.*

The critical segment $s_{a,b}$ is highlighted in orange in Figure 5.3.

Definition 5.6. *Consider an asynchronous chain* σ_a*. We denote* s_a^{header} *the* header segment of σ_a *defined as the subchain* $(\tau_a^1, \tau_a^2, \ldots, \tau_a^i)$ *where* $i \in [1, n_a - 1]$ *is the smallest integer such that* τ_a^{i+1} *has the lowest priority in* σ_a*. If* τ_a^1 *has the lowest priority then* s_a^{header} *is empty.*

In Figure 5.4, the header segment of σ_a is $(\tau_a^1, \tau_a^2, \tau_a^3, \tau_a^4, \tau_a^5)$ because τ_a^6 has the lowest priority in σ_a. In the same way, the header segment of σ_b is (τ_b^1). Definition 5.6 is useful for bounding the self-interfering in computing the busy-window of an asynchronous chain.

Definition 5.7. *Consider an asynchronous chain* σ_a*. If* σ_a *is deferred by* σ_b *then we denote* $s_{a,b}^{header}$ *the* header segment of σ_a w.r.t. σ_b *defined as the subchain* $(\tau_a^1, \tau_a^2, \ldots, \tau_a^i)$ *where* $i \in [1, n_a - 1]$ *is the smallest integer such that* τ_a^{i+1} *has lower priority than all tasks in* σ_b*.*

In Figure 5.4, the header segment of σ_a w.r.t. σ_b is $(\tau_a^1, \tau_a^2, \tau_a^3)$ because τ_a^4 has the priority 2, which is lower than all priorities of tasks in σ_b.

Definition 5.8. *If* σ_a *is deferred by* σ_b *then all segments in* σ_a *are deferred segments except* $s_{a,b}^{header}$ *when* σ_a *is asynchronous.*

We now revisit the worst-case latency analysis introduced in [105] and propose a description that is similar to worst-case response-time analysis as explained in [91].

[1] That is, if $i + l > n_a$ then it should be read $(i + l) \bmod n_a$.

Definition 5.9. *For a given task chain σ_b, a* σ_b-busy-window *is a maximal time interval $[t_1, t_2[$ where $\forall t \in [t_1, t_2[$ the resource is busy executing instances of σ_b and interfering tasks from other task chains.*

Definition 5.10. *The* q-instance busy time *of a chain σ_b is the maximum time it may take to process q instances of σ_b within a σ_b-busy-window starting with the first of these q instances.*

Theorem 5.1. *The q-instance busy time of $\sigma_b \in \mathcal{C}$ is bounded by*

$$\begin{aligned} B_b^+(q) \;:=\; & q \times C_b \\ & + \max(0, \eta_b^+(B_b^+(q)) - q) \times C_{s_b^{header}} \text{ if } \sigma_b \in \mathcal{AC} \\ & + \sum_{\sigma_a \in \mathcal{IC}(b)} \eta_a^+(B_b^+(q)) \times C_a \\ & + \sum_{\sigma_a \in \mathcal{AC} \cap \mathcal{DC}(b)} \eta_a^+(B_b^+(q)) \times C_{s_{a,b}^{header}} + \sum_{s \in S_a^b} C_s \\ & + \sum_{\sigma_a \in \mathcal{SC} \cap \mathcal{DC}(b)} C_{s_{a,b}} \end{aligned} \tag{5.1}$$

where C_x denotes the sum of the execution time bounds of the tasks in segment or chain x.

Proof. The above equation is made of five components:

1. The first line corresponds to the time needed to actually perform the q computations;
2. The second component accounts for the interference of additional instances of σ_b which may arrive while the q instances under consideration are being processed. Note that these instances will at most interfere until they have to execute the lowest priority task in σ_b. This component only applies to asynchronous chains;
3. The third element represents the interference from arbitrarily interfering chains, synchronous or asynchronous;
4. The fourth line deals with interference from deferred, asynchronous chains. Instances can arbitrarily queue up which allows the header segment to interfere arbitrarily. For all other segments at most one instance can be backlogged because tasks between segments have lower priority than tasks within segments. Each such instance can interfere for at most one segment (see below).

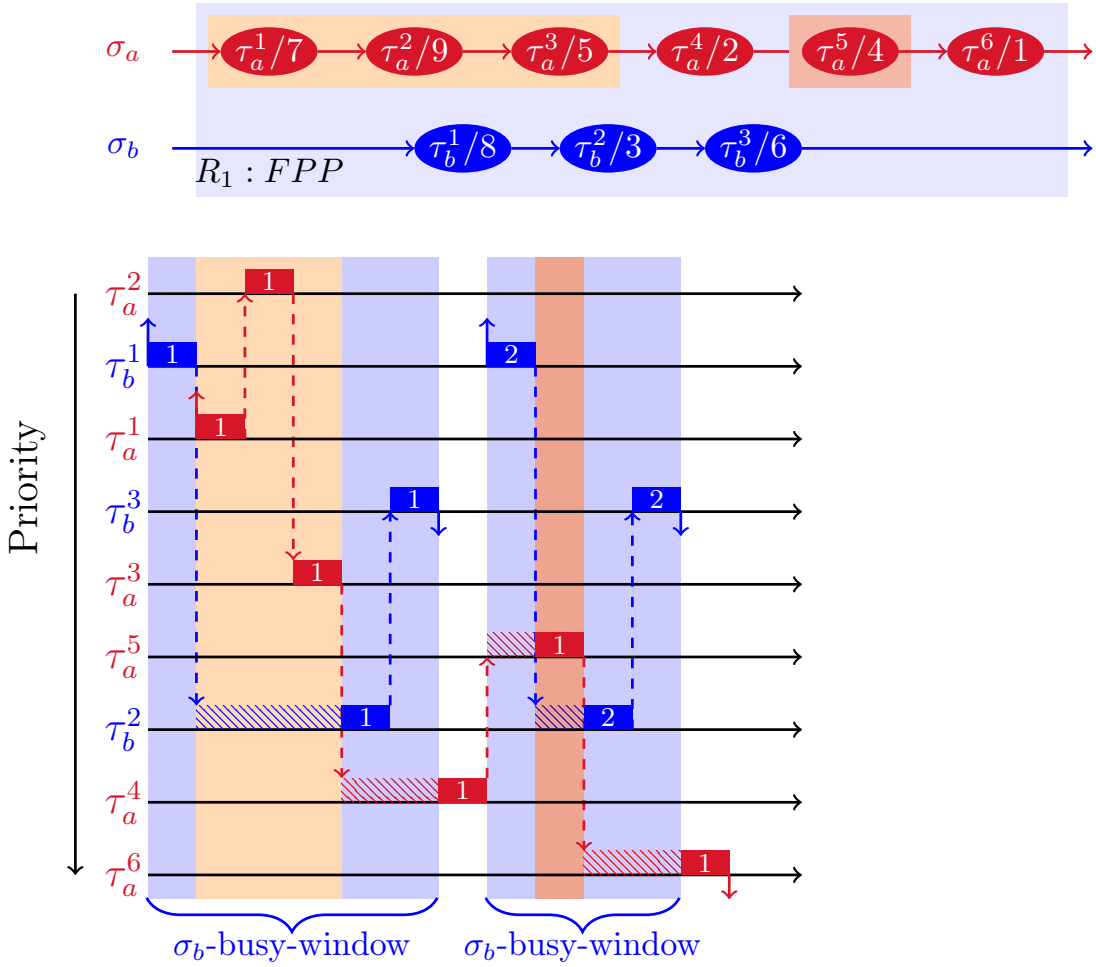

Figure 5.3: A σ_b-busy-window. Both σ_a and σ_b are synchronous task chains and σ_a is deferred by σ_b. The critical segment $s_{a,b}$ is highlighted in orange and the deferred segment $s = (\tau_a^5)$ is highlighted in vermilion.

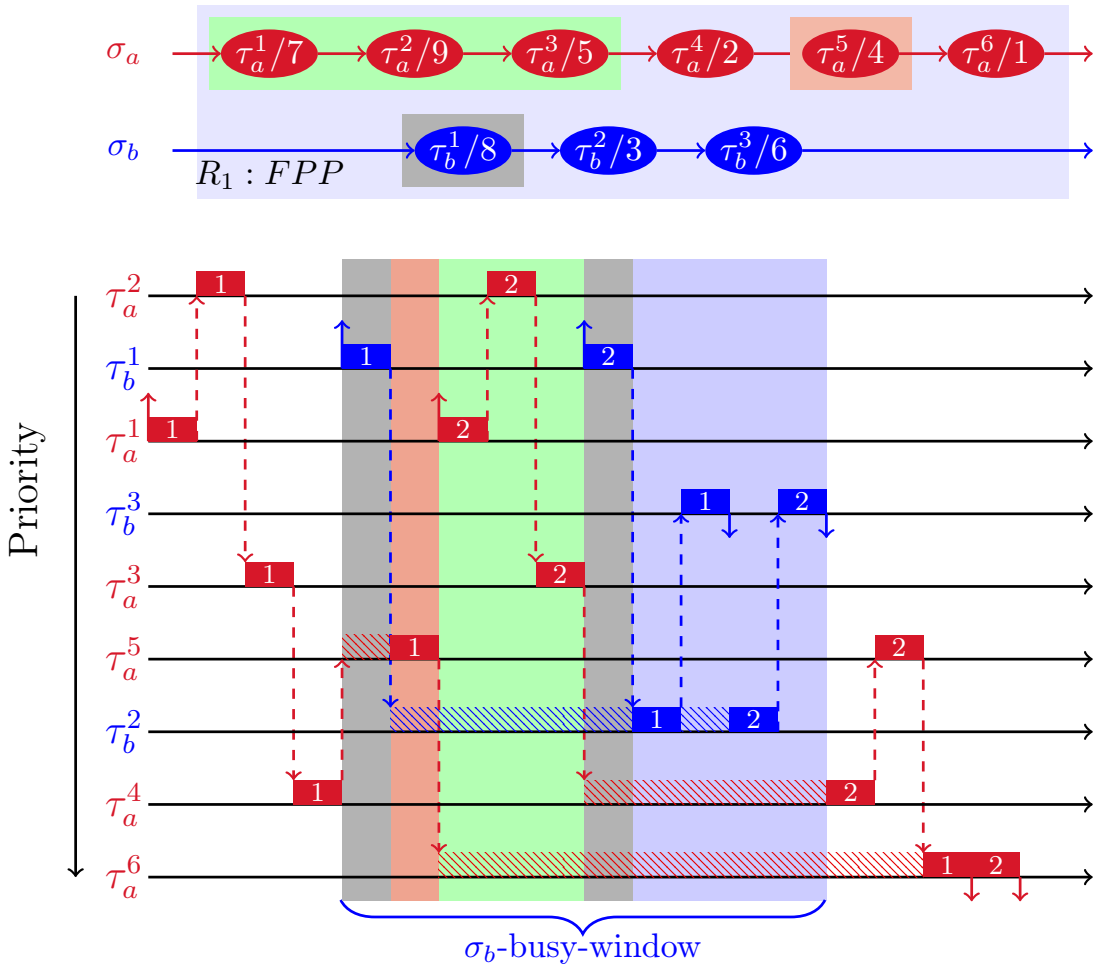

Figure 5.4: A σ_b-busy-window. Both σ_a and σ_b are asynchronous task chains and σ_a is deferred by σ_b. $s_{a,b}^{header}$, the header segment of σ_a w.r.t. σ_b, is highlighted in green and the deferred segment $s = (\tau_a^5)$ is highlighted in vermilion. s_b^{header}, the header segment of σ_b, is highlighted in grey.

5. The fifth component in the equation accounts for the interference from deferred, synchronous chains. Here only one instance per chain may interfere for at most one segment (see below). □

The correctness of the last two components in Equation (5.1) relies on the following property.

Lemma 5.1. *Tasks of a chain σ_a that are in different segments cannot execute instances corresponding to the same chain instance in the same σ_b-busy-window.*

Proof. Segments are maximal sequences of tasks with a priority higher than or equal to the lowest priority task, say τ_b^i, in σ_b. This means that between two segments of σ_a there is at least one task, say τ_a^j, that has lower priority than τ_b^i. In order to execute these two segments for the same instance of σ_a, one has to execute τ_a^j. Since τ_a^j has lower priority than all the tasks in σ_b, this can only happen after σ_b closes its current σ_b-busy-window. □

Figure 5.3 illustrates the σ_b-busy-window when both σ_a and σ_b are synchronous task chains, and Figure 5.4 illustrates it when both σ_a and σ_b are asynchronous task chains.

The maximum number of instances of σ_b in a σ_b-busy-window is

$$Q_b = \eta_b^+\{|BW_b^+|\} \tag{5.2}$$

where the length of the longest σ_b-busy-window is computed as follows:

$$\begin{aligned} |BW_b^+| \;:=\; & \eta_b^+(|BW_b^+|) \times C_b \\ & + \sum_{\sigma_a \in \mathcal{IC}(b)} \eta_a^+(|BW_b^+|) \times C_a \\ & + \sum_{\sigma_a \in \mathcal{AC} \cap \mathcal{DC}(b)} \eta_a^+(|BW_b^+|) \times C_{s_{a,b}^{header}} + \sum_{s \in S_a^b} C_s \\ & + \sum_{\sigma_a \in \mathcal{SC} \cap \mathcal{DC}(b)} C_{s_{a,b}} \end{aligned} \tag{5.3}$$

Theorem 5.2. *The latency of a task chain σ_b is bounded by*

$$WCL_b = \max_{1 \leqslant q \leqslant Q_b} \{B_b^+(q) - \delta_b^-(q)\} \tag{5.4}$$

Proof. This proof proceeds exactly as the proofs in Theorem 3.1.

Any q instances require at most $B_b^+(q)$ to finish, and the q-th instance arrives no earlier than $\delta_b^-(q)$ after the first instance in $B_b^+(q)$. Thus, the difference is an upper bound on the response time of the q-th instance of σ_b in all σ_b-busy-windows. Hence, the maximum of response times of instances in $B_b^+(Q_b)$ is the worst-case response time. □

5.3 Typical Worst-Case Analysis for Task Chains

The main objective of TWCA is to bound the number of deadlines-misses of a task chain σ_b, which may be caused by an instance at the input of an overload task chain σ_a. For that, we need to know over how many σ_b-busy-windows an instance of σ_a may span.

We already know that, in a chain σ_a, the execution of tasks corresponding to the same instance of σ_a cannot take place in the same σ_b-busy-window if those tasks are in different segments. This implies that an instance of σ_a spans over at least as many σ_b-busy-windows as there are segments of σ_a w.r.t. σ_b.

Note that there is no guarantee that a segment of σ_a will be executed within one σ_b-busy-window. It implies that Property 3.2 is not satisfied. We introduce the notion of *active segment*, which applies to subsegments that are guaranteed to be executed in the same σ_b-busy-window.

Definition 5.11. *An* active segment of σ_a w.r.t σ_b *satisfies:*

1. *it is a subchain*[2] *of a segment* $(\tau_a^i, \tau_a^{i+1}, \ldots, \tau_a^{i+k})$ *of* σ_a *where* $i \in [1, n_a]$ *and* $k \in [0, n_a - i]$*, i.e.,* $\forall l \in [0, k], \pi_a^{i+l} \geqslant \min_{1 \leqslant j \leqslant n_b}\{\pi_b^j\}$*; and*

2. *when* $k \geqslant 1$ *(the active segment contains more than one task):*

$$\forall l \in [1, k], \pi_a^{i+l} \geqslant \pi_b^{tail}$$

where τ_b^{tail} *denotes the tail task of* σ_b*.*

In other words, the active segment is the maximal subchain whose execution is guaranteed to not span over multiple σ_b-busy-windows. Note that the above definition implies that the active segment can be a single task τ_a^i with $\pi_a^i < \pi_b^{tail}$ because the second condition does not apply for a single-task active segment.

Example 5.2. *In Figure 5.2, chain* σ_a *has three active segments:* $(\tau_a^1, \tau_a^2), (\tau_a^3), (\tau_a^5)$*.* (τ_a^3) *is an active segment because* $\pi_a^3 > \pi_b^2$ *and the tail task of* σ_b *has no influence here.*

Lemma 5.2. *The execution of an active segment of* σ_a *w.r.t.* σ_b *cannot span over more than one* σ_b*-busy-window.*

Proof. Once the execution of an active segment of σ_a w.r.t. σ_b has started, τ_b^{tail} will not be able to execute because the active segment is blocking it or a task preceding it, and therefore the current σ_b-busy-window cannot be closed, until the whole segment □

[2]Here, $i + l$ is always smaller than or equal to n_a.

Note that an active segment is part of a segment in the sense of Definition 5.4. As a result, we easily conclude from Lemma 5.1 and 5.2 that two active segments of chain σ_a may be executed within one σ_b-busy-window if and only if they are part of the same segment of σ_a.

We now have all the ingredients needed to show how we extend TWCA to handle task chains. We follow here the same approach as the one for systems with independent tasks explained in Section 3.2. For the rest of the section we suppose given a chain σ_b and $k \geqslant 1$, and we focus on the computation of $dmm_b(k)$, that is, a bound on the number of deadlines that σ_b can miss out of a k-sequence, i.e., k consecutive instances. We assume that there is at most one instance of an overload chain σ_a in a σ_b-busy-window. As a result, we can without loss of generality consider our overload task chains as synchronous.

5.3.1 Deadline miss model computation

To compute a DMM, our strategy is again to

- compute an upper bound N_b on the number of deadlines that σ_b may miss within one busy-window;
- define $\tilde{\mathcal{C}}_b$, the set of unschedulable combinations w.r.t. σ_b;
- compute an upper bound $\Omega_k^{a \to b}$ on the number of instances of σ_a that could impact the considered k instances of σ_a.

Computation of N_b.

Lemma 5.3. *Let*

$$N_b := \#\{q \mid 1 \leqslant q \leqslant Q_b \wedge B_b(q) - \delta_b^-(q) > D_b\} \tag{5.5}$$

Then N_b is an upper bound on the number of deadlines that σ_b may miss within one busy-window.

Proof. The proof proceeds like that of Lemma 3.2. On the one hand, Q_b is the maximum number of instances in a σ_b-busy-window. On the other hand, response times of instances in $B_b^+(Q_b)$ upper bound the response times of instances in all σ_b-busy-windows. □

Computation of $\Omega_k^{j \to i}$.

Lemma 5.4. *Let*

$$\Omega_k^{a \to b} := \eta_a^+(\delta_b^+(k) + WCL_b) + 1 \tag{5.6}$$

Then $\Omega_k^{a \to b}$ is an upper bound on the number of instances of σ_a that could impact the considered k instances of σ_b.

Proof. Clearly, instances of chain σ_a that occur after the first instance of chain σ_b in the k-sequence is activated and before the last instance in the k-sequence finishes may have an impact on the latencies in the k-sequence. There are at most $\eta_a^+(\delta_b^+(k) + WCL_b)$ such instances. In contrast, an instance of σ_a that arrives after the last instance of chain σ_b in the k-sequence has finished does not impact the k-sequence. Finally, we have assumed that there is at most one instance of σ_a in a σ_b-busy-window so that at most one instance of σ_a before the k-sequence can impact it. □

5.3.2 Combinations for TWCA of task chains

For the case where tasks are independent, a *combination* is defined as a set of overload tasks. The DMM computation based on this definition heavily relies on the fact that one overload instance impacts exactly one busy-window. In the context of task chains, we have seen in the previous sections that one instance of a task chain σ_a may span over several σ_b-busy-windows. As a result, the impact of one overload instance is not limited here to one σ_b-busy-window. We have however also shown that the execution of an active segment of σ_a is restricted to a single σ_b-busy-window. Hence our choice to define combinations based on active segments rather than tasks or task chains.

Definition 5.12. *A* combination *$\bar{c}$ is a set of active segments w.r.t. σ_b such that if two active segments of the same chain σ_a are in $\bar{c}$ then they are part of the same segment of σ_a w.r.t. σ_b.*

Note that our definition excludes combinations that cannot execute within one σ_b-busy-window based on our definition of segment.

Example 5.3. *There are four possible combinations of the active segments of chain σ_a in Figure 5.2: $\{(\tau_a^1, \tau_a^2)\}, \{(\tau_a^3)\}, \{(\tau_a^5)\}, \{(\tau_a^1, \tau_a^2), (\tau_a^3)\}$.*

Definition 5.13. *A combination $\bar{c}$ is* schedulable *(w.r.t. σ_b) if σ_b is guaranteed not to miss any deadline in a σ_b-busy-window in which only the active segments in $\bar{c}$ execute (in addition to non-overload chains). Otherwise $\bar{c}$ is said to be* unschedulable.

5.3.2.1 Criterion of schedulability

We present here an efficient criterion to determine whether a combination $\bar{c}$ is in $\tilde{\mathcal{C}}_b$ or not. Let us reorganize Equation 5.1 for the multiple busy-time computation to show explicitly the contribution of the overload chains of a combination in the multiple busy time (and the latency) of σ_b.

$$\begin{aligned} B_b^{\bar{c}}(q) \;:=\; & q \times C_b \\ & + \max(0, \eta_b^+(B_b^{\bar{c}}(q)) - q) \times C_{s_b^{header}} \text{ if } \sigma_b \in \mathcal{AC} \\ & + \sum_{\sigma_a \in \mathcal{IC}(b) \setminus \mathcal{C}_{over}} \eta_a^+(B_b^{\bar{c}}(q)) \times C_a \\ & + \sum_{\sigma_a \in \mathcal{AC} \cap \mathcal{DC}(b)} \eta_a^+(B_b^{\bar{c}}(q)) \times C_{s_{a,b}^{header}} + \sum_{s \in S_a^b} C_s \\ & + \sum_{\sigma_a \in \mathcal{SC} \cap \mathcal{DC}(b) \setminus \mathcal{C}_{over}} C_{s_b^a} \\ & + \sum_{\sigma_a \in \mathcal{C}_{over}} \sum_{s \in \mathcal{S}_a} C_s \times r_s^{\bar{c}} \end{aligned} \tag{5.7}$$

where $r_s^{\bar{c}}$ is a Boolean, which holds exactly when $s \in \bar{c}$.

A combination $\bar{c}$ is schedulable if $B_b^{\bar{c}}(q) - \delta_b^-(q) \leqslant D_b$ for all $q \in [1, Q_b]$. Now, let us define $L_b(q)$ as follows.

$$\begin{aligned} L_b(q) := \; & q \times C_b \\ & + \max(0, \eta_b^+(\delta_b^-(q) + D_b) - q) \times C_{s_b^{header}} \text{ if } \sigma_b \in \mathcal{AC} \\ & + \sum_{\sigma_a \in \mathcal{IC}(b) \setminus \mathcal{C}_{over}} \eta_a^+(\delta_b^-(q) + D_b) \times C_a \\ & + \sum_{\sigma_a \in \mathcal{AC} \cap \mathcal{DC}(b)} \eta_a^+(\delta_b^-(q) + D_b) \times C_{s_{a,b}^{header}} + \sum_{s \in S_a^b} C_s \\ & + \sum_{\sigma_a \in \mathcal{SC} \cap \mathcal{DC}(b) \setminus \mathcal{C}_{over}} C_{s_b^a} \end{aligned} \tag{5.8}$$

Then we now have a much simpler sufficient condition for schedulability: $\bar{c}$ is schedulable if

$$\forall q \in [1, Q_b],\; L_b(q) + \sum_{\sigma_a \in \mathcal{C}_{over}} \sum_{s \in \mathcal{S}_a} C_s \times r_s^{\bar{c}} \leqslant \delta_b^-(q) + D_b \tag{5.9}$$

Consequently, one can deduce that the superset of unschedulable combinations w.r.t. σ_b is then:

$$\tilde{\mathcal{C}}_b := \{\bar{c} \mid \exists q \in [1, Q_b],\; L_b(q) + \sum_{\sigma_a \in \mathcal{C}_{over}} \sum_{s \in \mathcal{S}_a} C_s \times r_s^{\bar{c}} > \delta_b^-(q) + D_b\} \tag{5.10}$$

5.3.3 An ILP formulation for the DMM

Theorem 5.3. *Let us define* $dmm_b(k)$ *as*

$$dmm_b(k) := \max \left\{ N_b \sum_{\bar{c} \in \tilde{\mathcal{C}}_b} x_{\bar{c}} \,|\, \forall \sigma_a \in \mathcal{C}_{over}, \forall s \in \mathcal{S}_a, \sum_{\{\bar{c} \in \tilde{\mathcal{C}}_b \,|\, s \in \bar{c}\}} x_{\bar{c}} \leqslant \Omega_k^{a \to b} \right\} \tag{5.11}$$

where

- $x_{\bar{c}}$ *is the variable constraining the number of busy-windows that could contain one instance of the* k*-sequence and suffer from an overload corresponding to* $\bar{c} \in \tilde{\mathcal{C}}_b$*;*
- $\mathcal{S}_a$ *denotes the set of active segments of* σ_a*.*

Then $dmm_b(k)$ *is a DMM for* σ_b*.*

Proof. Assume that we have $\Omega_k^{a \to b}$ for all chains σ_a. In the worst case, each active segment of σ_a also impacts σ_b $\Omega_k^{a \to b}$ times. As in Section 3.2, we here also face a multi-dimensional knapsack problem where items correspond to unschedulable combinations and capacities to $\Omega_k^{a \to b}$ for every line s associated with an active segment of overload chain σ_a. So considering that $x_{\bar{c}}$ stands for the number of times that a combination $\bar{c}$ is used in the packing under consideration, we want to find the packing that maximizes the number of deadline-misses of σ_b — which is equal to the number of unschedulable combinations used multiplied by the maximum number of deadline-misses due to each combination. This packing is constrained by the fact that active segments cannot be used in more combinations than is allowed by their corresponding $\Omega_k^{a \to b}$. □

We have now shown how we can reuse the ILP solution in Equation 3.19 for systems with task chains with limited changes.

5.4 Experiments

5.4.1 Industrial case study

We first present the experiments performed based on the TRT case study, which is illustrated in Figure 5.1. Remember that the system is a single-core processor provided with an FPP scheduler. Four task chains share the resource, namely $\sigma_a, \sigma_b, \sigma_c$ and σ_d, where σ_a and σ_b are overload chains. In the following experiments we focus on providing DMMs for σ_c and σ_d.

The worst-case latency WCL of task chains σ_c and σ_d was computed as described in Section 5.2. The analysis results show that the system is not schedulable as σ_c can in the worst-case miss its deadline, see Table 5.1.

task chain	WCL	D
σ_c	331	200
σ_d	175	200

Table 5.1: WCL of task chains σ_c and σ_d [49].

A second analysis, in which all overload chains are abstracted away, reveals that the system is schedulable and σ_c meets its deadline if neither σ_a nor σ_b are activated. We thus perform TWCA as presented in this chapter. The computed DMM of σ_c is shown in Table 5.2 — σ_d is schedulable and therefore does not need a DMM.

task chain	DMM
σ_c	$dmm_c(3) = 3$, $dmm_c(76) = 4$, $dmm_c(250) = 5$

Table 5.2: $dmm(k)$ for task chain σ_c [49].

Let us provide additional details resulting from this DMM computation. Both chains σ_a and σ_b arbitrarily interfere with σ_c because neither has a task with a priority lower than 1, which is the lowest priority in σ_c. As a result σ_a and σ_b have only one segment, respectively (τ_a^1, τ_a^2) and $(\tau_b^1, \tau_b^2, \tau_b^3)$. These two segments are also active segments because the priority of the tail task of chain σ_c is lower than all priorities in these segments (see Figure 5.1). Therefore, no constraints on combining active segments are needed. The set of combinations thus has three elements: $\bar{c}_1 = \{(\tau_a^1, \tau_a^2)\}$, $\bar{c}_2 = \{(\tau_b^1, \tau_b^2, \tau_b^3)\}$, and $\bar{c}_3 = \{(\tau_a^1, \tau_a^2), (\tau_b^1, \tau_b^2, \tau_b^3)\}$. Based on the schedulability criterion introduced in the previous section , $\bar{c}_3$ is the only unschedulable combination, so in this case the TWCA is fairly simple.

To generalize the results obtained on the industrial case study, while preserving practical relevance, we arbitrarily modify the priority assignment to generate random systems with different scenarios.

5.4.2 Synthetic test cases

We arbitrarily assign priorities to show the impact of priority assignments on the schedulability and the deadline miss models. In this experiment we randomly choose 1000 assignments to test our analysis intensively. Figure 5.5 shows $dmm_c(10)$ and $dmm_d(10)$. Notice first that out of the 1000 assignments generated,

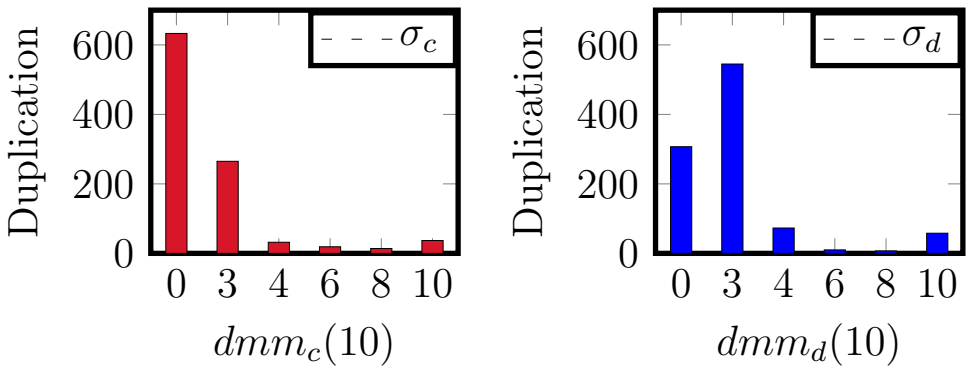

Figure 5.5: $dmm_c(10)$ and $dmm_d(10)$ [49].

chain σ_c is schedulable (misses no deadline) 633 times. More interestingly, chain σ_d is schedulable only 307 times out of 1000. TWCA in that case is very useful as for more than 500 of the remaining systems it can guarantee that no more than 3 out 10 deadlines can be missed. Note that we have repeated our experiment 30 times and observed similar results.

5.5 Related Work

To the best of our knowledge, there is no state-of-the-art method for the computation of weakly-hard guarantees in real-time systems with task dependencies.

Extensive research has focused on the schedulability analysis of *hard* real-time systems with task dependencies. This includes approaches focusing on offset analysis [44] but also more general precedence models [45]. In [105], Schlatow and Ernst presented an upper bound on the end-to-end latency of task chains in real-time systems, on which we base our work in this chapter. Later in [106], they extended the analysis to consider complex precedence and blocking relations. Recently, Girault et al. presented in [42] an analysis to compute upper and lower bounds on the worst-case latency to estimate the tightness of the WCL. Their analysis distinguishes between *header*, *inner* and *tail* segments, and it investigates the execution dependencies between interfering task chains. Therefore, they showed that the computed upper bound on WCL is much tighter than the one computed in [105]. Computing lower bounds on WCL would be a good tool to investigate the pessimism of TWCA.

In contrast, there is little in the literature regarding the analysis of WHRT systems (see Section 3.5 for related work). However, non of these works can handle task dependencies.

5.6 Summary

In this chapter, we presented an extension of TWCA that is the first method for computing end-to-end deadline miss models for systems with task dependencies. This bounds the number of potential deadline-misses in a given sequence of instances of a task chain. Our approach addresses uniprocessor systems with FPP scheduling. We show how state-of-the-art TWCA can be extended using recent results in the analysis of hard real-time systems with task dependencies. Specifically, we show how we can formulate our problem as a knapsack problem. Our approach is validated on a realistic case study inspired by industrial practice and synthetic variants of it. This work with the case study has been demonstrated at RTAS 2017 [55].

6 BUDGETING UNDER-SPECIFIED TASKS IN WEAKLY-HARD REAL-TIME SYSTEMS

Quantifying the time budget available to add security, monitoring or recovery tasks such that the system still meets its timing constraints is an interesting problem in the design of real-time systems [127]. In WHRT systems, specifying parameters of any extra task, e.g., recovery task, has to take into consideration the (m, k) constraints of system tasks. This is the problem addressed in this chapter, as motivated by a case study inspired by industrial practice in the course of the collaboration project between TU Braunschweig and TRT. Thales Alenia Space (TAS) [114] provided the considered case study.

In this work, we present an extension of slack analysis [28] for budgeting in the design of WHRT systems. Slack analysis can help to anticipate the impact of new tasks on the system schedulability. The case study will be used to motivate and validate the proposed analysis. The four main contributions of this chapter are:

- an extension of slack analysis to compute the maximum slack, which guarantees that no more than m deadline-misses out of k consecutive executions can happen;
- an execution time budget for under-specified tasks based on the multiframe task model [77] where we consider that each under-specified task has two execution times: a long one and short one;
- a methodology that explains why and how this method should be used safely in the design of systems that have hard and weakly-hard requirements;
- a case study dealing with satellite on-board software, which shows the practical usefulness of weakly-hard constraints and how to guarantee them.

The contribution in this chapter is the second extension to the core contribution of this thesis and it has been published in [52].

This chapter is organized as follows: first we present a case study provided by TRT as a motivation for this extension, and we formulate the addressed problem in Section 6.1. Section 6.2 shows how to budget the under-specified tasks to satisfy hard real-time constraints. We present in Section 6.3 an extension of slack analysis and our general approach for budgeting based on the multiframe task model. In Section 6.4 we summarize the methodology that we propose. We present our experimental results in Section 6.5 and discuss related work in Section 6.6. Section 6.7 concludes.

6.1 Case Study and Problem Statement

In this section we introduce the case study that motivates the work presented in this chapter.

6.1.1 Satellite on-board software

A satellite comprises two main parts: the platform and the payload. The payload realizes the main satellite mission. The payload is typically characterized by high computation requirements but in the general case its software is considered at best firm or soft real-time.

The platform is the service module that governs the satellite and ensures the execution of the mission. The platform on-board software (OBSW) implements all major functions of the satellite.

A subset of those OBSW functions are characterized by hard real-time requirements. In contrast, some tasks executing some less critical functions, may occasionally miss deadlines without catastrophic consequences on the mission, and at most some performance degradation. One example is the Attitude and Orbit Control System (AOCS) functions, where sensor acquisition and processing are somehow robust to occasional deadline-misses because of the intrinsic robustness of the implemented control laws.

The OBSW is however traditionally designed, analyzed and implemented with techniques typical of safety-critical, hard real-time systems. This implies that all tasks defined for the OBSW are considered as hard real-time and treated as such in the schedulability analysis used to confirm the system feasibility. The analysis is performed using representative worst-case operational scenarios. The reason for this choice is twofold:

1. It is much easier to prove to clients that the system is schedulable and fulfills the mission goals by treating all tasks as hard real-time, with a design

process and analysis equations consolidated along several years, and without admitting exceptions on the treatment of task deadlines.

2. The OBSW development team does not know completely the possible consequences of deadline-misses from the point of view of performance degradation or function losses, as such knowledge requires deep analysis at system / avionics level. It is therefore not obvious to understand if deadline-misses are admissible in the overall mission context.

Current satellite OBSW is typically executed on a single-core processor using an FPP scheduling policy. Considering the tasks as hard real-time leads to a resource *over-provisioning*. The standard for space software engineering [34] specifies the utilization margin of the resource $(1 - U)$ with 25% for Critical Design Review (CDR). By relaxing the constraints to WHRT, we aim to reduce the over-provisioning.

Table 6.1 shows a representative task set and the real-time attributes of each task. The attributes are representative of a high-load scenario for the OBSW in a mission operational mode. Each task τ_i in the given system is characterized by its:

- priority index π_i; for simplicity of notation, we assume that tasks are given in order of their static priority, i.e., τ_j has higher priority than τ_i for every $j < i$;
- type of task release pattern: periodic (P), software sporadic (S), hardware sporadic (HWS), i.e., triggered by an interrupt, background task;
- worst-case execution time C_i; this value is not based on static analysis but rather on the observed execution times;
- period or interarrival time T_i;
- relative deadline D_i — all deadlines are constrained ($D_i \leqslant \delta_i^-(2)$);

Note that some tasks specified as sporadic have in fact a pseudo-periodic behavior.

Table 6.1 includes two different kinds of tasks:

(i) *nominal tasks*: tasks that are active and executed in the represented operational scenario;

(ii) *recovery tasks*: tasks that are involved in asynchronous fault handling or recovery activities and are triggered only on given fault / error occurrences. They are marked as gray in the table.

Name	$\boldsymbol{\pi}$	**Type**	$\boldsymbol{C}$	$\boldsymbol{T}$	$\boldsymbol{D}$
τ_1	1	HWS	0.56	15.625	15.625
τ_2	2	P	0.76	15.625	15.625
τ_3	3	P	15	125	31.25
τ_4	4	P	25.03	125	46.875
τ_5	5	P	7.5	62.5	62.5
τ_6	6	P	6.15	125	125
τ_7	7	P	1.2	125	125
τ_8	8	P	0.9	1000	500
τ_9	9	P	1.95	250	250
τ_{10}	10	S		10000	125
τ_{11}	11	S			125
τ_{12}	12	P	1.2	125	125
τ_{13}	13	P	5.15	250	203.125
τ_{14}	14	P	1.2	1000	500
τ_{15}	15	P	22.5	500	500
τ_{16}	16	P	3.5	250	250
τ_{17}	17	P	27	500	500
τ_{18}	18	HWS	1.5	1000	1000
τ_{19}	19	P	16	1000	1000
τ_{20}	20	P	19.1	1000	1000
τ_{21}	21	S			1000
τ_{22}	22	P	88.8	2000	2000
τ_{23}	23	P	2	32000	32000
τ_{24}	24	P	1	32000	32000
τ_{25}	25	P	1	1000	1000
τ_{26}	26	S	20	1000	1000
τ_{27}	27	S	40	2000	2000
τ_{28}	28	S	1.5	2000	2000
τ_{29}	29	S	1.5	2000	2000
τ_{30}	30	P	0.2	32000	32000

Table 6.1: A task set representative of OBSW. $\boldsymbol{\pi}$, $\boldsymbol{C}$, $\boldsymbol{T}$, and $\boldsymbol{D}$ denote respectively: priority, worst-case execution time, period/minimum distance, deadline. The time unit is ms.

Among the nominal tasks, some have real-time constraints that we will consider as hard real-time; others can be considered as WHRT, as they can withstand occasional deadline-misses without significant system-level consequences.

6.1.2 Problem statement

The specification of recovery tasks typically occurs in the latest development phases, and therefore their characteristics are not known until late in the development cycle. The execution of such recovery tasks may however perturb the execution of nominal tasks, leading to deadline-misses that would potentially induce a degradation of the system performance.

Configuring the timing attributes of the recovery tasks represents a challenging timing issue for the real-time architect. It is important to guarantee that the reconfiguration and recovery tasks can accomplish their functions, which are related to the safety of the spacecraft. At the same time, it is necessary to preserve a sound timing behavior for the nominal tasks.

Moreover, the timing behavior of the recovery tasks must be established and assessed as early as possible in the development, as in later phases the development of the rest of the software system approaches completion, with little freedom for significant modifications.

The reader should note that the problem statement regards finding a convenient method for assigning attributes and guarantees to such tasks, rather than establishing a complete fault tolerance strategy [102][75] for the on-board software and the satellite. The latter requires a much more global reasoning at system level and it is not in the scope of this work. There are several mechanisms (hardware and software) that are devoted to the implementation of such global fault tolerance strategy for a given satellite, and the method we seek would simply concur to their definition in a convenient manner.

To solve this problem, there is a need for a timing verification method fulfilling two conditions:

(i) applicability at early design stages;

(ii) a guarantee on the provided upper bounds for the tasks' response times.

Worst-case response time analysis seems well adapted to solve the timing challenge mentioned above, since its applicability already starts with the early conceptual design phases and it provides formal proofs based on a mathematical model of the system timing behavior. These proofs allow calculating safe lower and upper bounds on the response times, thus guaranteeing corner-case coverage.

Classic worst-case response-time analysis would however not be able to take into account the weakly-hard nature of some tasks, and would just check that deadlines of those tasks are met in the worst case. This would lead to an under-estimation of the timing budget available for the recovery tasks (and therefore to ensure the safety functions), which could be delicate, especially in case of a system with a high CPU load.

The real-time architect needs a method that takes into account the weakly-hard nature of some tasks, and can provide him/her with means to perform tradeoffs on the budget to be assigned to the recovery tasks. Such a method would be considered attractive in this context.

Let us now formulate the problem. We consider a single processing resource that schedules a task set $\mathcal{Z} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ according to the FPP scheduling policy. Each task $\tau_i \in \mathcal{Z}$ is modeled by its

- worst-case execution time C_i
- activation model described using arrival curves
- priority π_i
- constrained relative deadline $D_i \leqslant \delta_i^-(2)$

The task set $\mathcal{Z}$ is partitioned into *nominal tasks*, which are fully specified, and *recovery tasks*, for which only priorities and deadlines are known, such that we call these *under-specified tasks*. We denote by $\mathcal{N}$ the set of nominal tasks and $\mathcal{R}$ the set of under-specified tasks. Under-specified tasks are considered to be sporadic. WHRT constraints are assumed to be given for nominal tasks.

Our problem is to provide a set of constraints on the execution times and the activation patterns of the tasks in $\mathcal{R}$ that is sufficient (and ideally necessary too) to guarantee (m, k)-schedulability of all tasks in $\mathcal{N}$, where a task is said to be (m, k)-*schedulable* if it misses no more than m deadlines out of a sequence of k consecutive executions.

6.2 Budgeting with Hard Real-Time Constraints

The problem in this section is to provide a set of constraints on the load incurred by the tasks in $\mathcal{R}$ (i.e., recovery tasks, a.k.a. under-specified tasks) that is sufficient to guarantee schedulability of all tasks in the nominal mode.

This section assumes that the reader is familiar with the worst-case response time analysis of FPP, which is elaborated in Section 3.2.2. Namely, the reader

should be aware of the concept of τ_i busy-window. However, the needed equations are listed here for better readability:

- The multiple activation busy time, see Eq. 3.1

$$B_i^+(q) = q \;.\; C_i + \sum_{\tau_j \in hp(i)} \eta_j^+(B_i^+(q)) \;.\; C_j.$$

- The maximum number of instances of τ_i in a τ_i busy-window, see Eq. 3.3

$$Q_i = \min\{q \geqslant 1 \mid B_i^+(q) < \delta_i^-(q+1)\}.$$

- The length of the longest τ_i busy-window, see Eq. 3.4

$$|BW_i^+| = B_i^+(Q_i).$$

- The response time of every instance of τ_i, see Eq. 3.11

$$R_i^q = B_i^+(q) - \delta_i^-(q).$$

- The response time of τ_i, see Eq. 3.12

$$R_i^+ = \max_{1 \leqslant q \leqslant Q_i} \{R_i^q\}.$$

Let us first recall some results related to slack analysis [28, 66, 98, 117].

Definition 6.1. *The* slack S_i^0 *of task* τ_i *is the maximum amount of processing time available to an instance of* τ_i *that may be stolen from any instance of* τ_i *without causing its deadline to be missed.*

The slack of a task τ_i can be computed by noticing that any τ_i idle-window between the completion of an instance of τ_i and its deadline can be used for computation of that instance without causing it to miss its deadline.

Definition 6.2. *By* τ_i idle-window *we refer to any maximal time interval between two* τ_i *busy-windows.*

Theorem 6.1. *For FPP scheduling, the slack of* τ_i *is equal to the sum of all* τ_i *idle-windows between the critical instant and* D_i *in the worst-case busy-window.*

This is illustrated in Figure 6.1.

Let us now focus on a task τ_i in the nominal mode. Denote $\mathcal{R}_i$ the set of under-specified tasks with a priority higher than τ_i. We can directly reuse the concept of slack to budget the under-specified tasks.

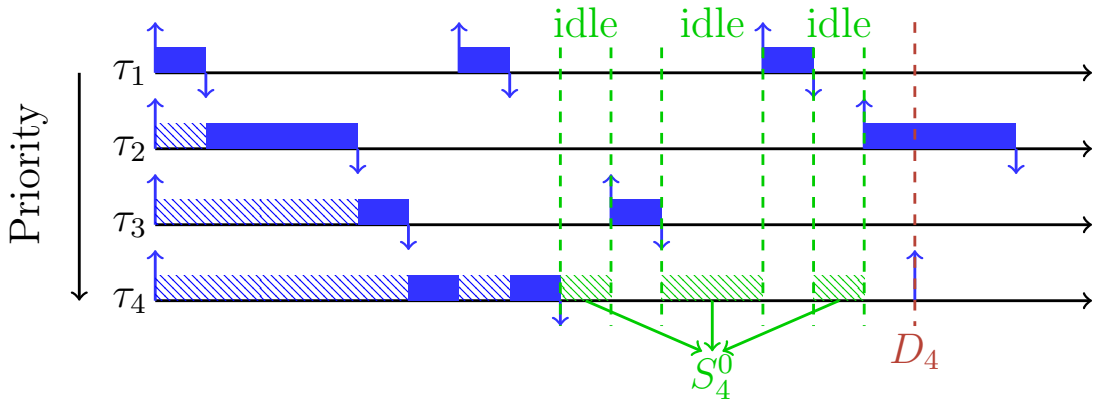

Figure 6.1: Worst-case busy-window analysis. The slack S_4^0 of τ_i is shown.

Lemma 6.1. *Let S_i^0 be the slack of τ_i in the system made of only nominal tasks (i.e., excluding under-specified tasks). If 1) $\sum_{\tau_r \in \mathcal{R}_i} C_r \leqslant S_i^0$ and 2) $\forall \tau_r \in \mathcal{R}_i$: $\delta_r^-(2) > D_i$ then τ_i is schedulable.*

Proof. This follows directly from the definition of slack. Note that we need to ensure that at most one instance of any under-specified task will interfere with a given instance of τ_i for the result to hold. □

We now relax the second constraint in the above lemma to consider multiple instances of an under-specified task in one busy-window.

Lemma 6.2. *Let BW_i^0 be the longest τ_i busy-window obtained by analyzing the nominal task set with an additional load of size S_i^0. That is:*

$$BW_i^0 := \min\{\Delta T \geqslant 0 \mid \Delta T = C_i + S_i^0 + \sum_{\tau_j \in \mathcal{N} \cap hp(i)} \eta_j^+(\Delta T) \times C_j\}$$

If $\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(BW_i^0) \times C_r \leqslant S_i^0$ then τ_i is schedulable.

Proof. Again, this follows directly from the definition of slack. In this case the slack used by an under-specified task τ_r is shared among several of its instances. □

We can now state our general result on how to budget under-specified tasks to guarantee hard real-time schedulability of all nominal tasks.

Theorem 6.2. *If for all $\tau_i \in \mathcal{N}$*

$$\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(BW_i^0) \times C_r \leqslant S_i^0 \tag{6.1}$$

then the system is schedulable.

Proof. The above equation and Lemma 6.2 guarantee together that all nominal tasks remain schedulable in presence of under-specified tasks satisfying the given constraints. □

If this budget is acceptable then there is no need to consider budgeting for the weakly-hard case. The rest of this chapter is dedicated to proposing solutions if a larger budget is needed for execution times of the under-specified tasks.

6.3 Budgeting with Weakly-Hard Real-Time Constraints

The problem is now to provide a set of constraints on the load incurred by the tasks in $\mathcal{R}$ that is sufficient to guarantee *weakly-hard* schedulability, see Definition 2.25, of all tasks in the nominal mode rather than (hard) schedulability.

Again, we first focus on a task τ_i in the nominal mode, this time supposing that it has an (m, k) constraint, i.e., τ_i may miss no more than m out of k deadlines. Denote $\mathcal{R}_i$ the set of under-specified tasks with a priority higher than τ_i.

Section 3.2 shows that the standard way to establish (m, k)-schedulability using TWCA is to consider a sequence of k consecutive instances of τ_i and to prove that no more than m instances in this sequence may miss their deadline. In the case of this work the instances of under-specified tasks can be considered as overload since they are not taken into account by the initial worst-case analysis. We can therefore adapt TWCA to our context. We reuse in particular the following notations.

- N_i, the number of deadline-misses that occur in the longest τ_i busy-window BW_i^+ of the system with nominal and under-specified tasks, see Definition 3.2.
- $\Omega_k^{r\to i}$, the maximum number of instances of higher-priority under-specified task τ_r that may occur within a window of k-sequence, see Definition 3.3.

  $$\Omega_k^{r\to i} = \eta_r^+(BW_i^+ + \delta_i^+(k) + R_i^+)$$

  and Ω_i the sum over all higher-priority under-specified tasks:

  $$\Omega_i = \sum_{\tau_r \in \mathcal{R}_i} \Omega_k^{r\to i}$$

Let ΔT_k^i denotes the time window of k-sequence:

$$\Delta T_k^i = BW_i^+ + \delta_i^+(k) + R_i^+$$

Hence, $\Omega_k^{r \to i} = \eta_r^+(\Delta T_k^i)$.

Notice here that budgeting according to constraints on N_i and ΔT_i^k is not easy as these parameters themselves depend on the parameters of the under-specified tasks. In the next section we first focus on how to relate the load budget of recovery tasks and N_i, i.e., the maximum number of deadline-misses in a single busy-window.

6.3.1 Extending the concept of slack to weakly-hard systems

Let us start with a few lemmas.

Lemma 6.3. *There can be more than one instance of a given task τ_i in one τ_i busy-window only if that task misses its deadline in that busy-window. Formally: $Q_i \geqslant 2$ only if $R_i^+ > D_i$.*

Proof. By definition of Q_i, $B_i^+(Q_i) \leqslant \delta_i^-(Q_i + 1)$ and for any $q < Q_i$, $B_i^+(q) > \delta_i^-(q + 1)$. For $q = 1 : B_i^+(1) > \delta_i^-(2)$. We work with constrained deadlines so $D_i \leqslant \delta_i^-(2)$ so $B_i^+(1) > D_i$. As $B_i^+(1) = R_i^1$ and therefore $R_i^+ \geqslant B_i^+(1)$ we can conclude that $R_i^+ > D_i$. □

This lemma is easily generalized to consecutive deadline-misses:

$$\forall q < Q_i, R_i^q > D_i.$$

This result is useful for us as it directly relates the number of deadline-misses in a busy-window with the length of that busy-window. In particular, we obtain that $N_i = Q_i - 1$ if $B_i^+(Q_i) \leqslant \delta_i^-(Q_i) + D_i$.

Let us now go one step further and extend the slack analysis of Section 6.2 to systems in which a bounded number of deadline-misses are allowed.

Definition 6.3. *For $\mu \in \mathbb{N}$, the μ-slack of a task τ_i, denoted S_i^μ, is the maximum amount of processing time that may be stolen from τ_i in a τ_i busy-window without causing more than μ deadlines of τ_i to be missed in a row.*

The μ-slack of a task τ_i can be computed in a way similar to the usual slack but focusing on the $(\mu + 1)$-th deadline instead of the first deadline.

Theorem 6.3. *For FPP scheduling, the μ-slack of τ_i is equal to the sum of all τ_i idle-windows between the critical instant and $\delta_i^-(\mu + 1) + D_i$ in the worst-case busy-window.*

Proof. The above condition guarantees that the $(\mu + 1)$-th deadline is met. □

Let us now introduce a definition, which will be useful to bound BW_i^+ and R_i^+.

Definition 6.4. *Let BW_i^μ be the longest τ_i busy-window obtained by analyzing the nominal task set with an additional load of size S_i^μ. We know that such a busy window contains exactly $\mu+1$ instances of τ_i so:*

$$BW_i^\mu := \min\{\Delta T \geqslant 0 \mid \Delta T = (\mu+1) \times C_i + S_i^\mu + \sum_{\tau_j \in \mathcal{N} \cap hp(i)} \eta_j^+(\Delta T) \times C_j\}$$

Since τ_i may not miss more than m deadlines in a row, we can conclude that $BW_i^+ \leqslant BW_i^m$. Similarly, R_i^+ is bounded by the response times of τ_i observed in BW_i^m. We thus know how to define ΔT_i^k. Let us now state the condition which guarantees that τ_i may not miss more than m deadlines in a row, and thus $N_i = m$.

Lemma 6.4. *If $\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(BW_i^m) \times C_r \leqslant S_i^m$ then τ_i cannot miss more than m deadlines in a row.*

Proof. This is a direct consequence of the definition of m-slack. □

At this point, it may seem that the intuitive, if pessimistic, way to budget the under-specified tasks is to require that $\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(\Delta T_i^k) \times C_r \leqslant S_i^m$. This, however, is not a sufficient condition for (m,k)-schedulability. The reason is that the same load incurred by under-specified tasks may result in more deadline-misses if they happen in different busy-windows. This is the meaning of the following lemma.

Lemma 6.5.

$$\forall \mu \in \mathbb{N}^+ : S_i^\mu \geqslant (\mu+1) \times S_i^0$$

Proof. Consider a sequence of $\mu+1$ consecutive instances of τ_i. Remember that S_i^0 is the sum of all τ_i idle-windows between the critical instant and D_i in the worst-case busy-window. Because deadlines are constrained, this is smaller than or equal to the sum of all τ_i idle-windows between the critical instant and $\delta_i^-(2)$ in the worst-case busy-window. Allowing only S_i^0 slack for each instance in the sequence furthermore assumes that the critical instant may repeat for each instance, which is pessimistic compared to the way S_i^μ is computed. As a result, S_i^μ provides more slack than $(\mu+1) \times S_i^0$. □

The consequence of this is that a safe bound on the budget for the under-specified tasks must be based for now on S_i^0.

Lemma 6.6. *Let $\Lambda_i = (m+1) \times S_i^0$. If*

$$\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(\Delta T_i^k) \times C_r \leqslant \Lambda_i$$

then τ_i is (m,k)-schedulable.

Proof. We have to prove that a load of Λ_i within ΔT_i^k causes no more than m consecutive deadline-misses if it occurs in one τ_i busy-window, and no more than m non-consecutive deadline-misses if it distributes over several busy-windows.

- The first condition is directly satisfied by Lemmas 6.4 and 6.5.
- Suppose now that Λ_i is distributed over n τ_i busy-windows with l_b denoting the load in each busy-window: $\sum_{b=1}^{n} l_b = \Lambda_i$. For each l_b let μ_b denote the maximum number of (consecutive) deadline-misses that may be caused by l_b ($\mu_b \geqslant 0$). We have to prove that $\sum_{b=1}^{n} \mu_b \leqslant m$. By definition we know that $l_b > S_i^{\mu_b - 1}$ for all l_b so from Lemma 6.5 we can derive that $l_b > \mu_b \times S_i^0$. If we now sum this over all l_b we get

$$\sum_{b=1}^{n} l_b > \sum_{b=1}^{n} \mu_b \times S_i^0$$

  Since $\sum_{b=1}^{n} l_b = \Lambda_i = (m+1) \times S_i^0$ we can conclude that $m + 1 > \sum_{b=1}^{n} \mu_b$, which is what we had to prove.

□

Theorem 6.4. *If for all $\tau_i \in \mathcal{N}$ with an (m, k) schedulability constraint*

$$\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(\Delta T_i^k) \times C_r \leqslant (m+1) \times S_i^0 \tag{6.2}$$

then the system satisfies its hard and weakly-hard requirements.

Proof. This results is a direct consequence of Lemma 6.6. □

This result is obviously quite pessimistic. It is clear at this point that obtaining better bounds requires us to use a more fine-grained model of how load distributes over busy-windows. We investigate this possibility in the next section.

6.3.2 Budgeting for multiframe tasks

Here, we focus on a specific application scenario and assume that each under-specified task performs two activities:

- A frequent monitoring activity with a relatively short execution time aiming at analyzing deviations from safe state in the system and perform some rapid recovery or triggering higher-level recovery, characterized by a short minimum distance between two consecutive occurrences.

- A less frequent failure recovery activity (e.g., an avionics reconfiguration procedure), which requires a longer execution time and characterized by a longer minimum time distance between two consecutive executions.

Based on the behavior described above, the execution time model of any under-specified task τ_r can be characterized by (C_r^l, C_r^s, x) where:

- C_r^s is the short execution time corresponding to the recovery activity of the task;
- C_r^l is the long execution time corresponding to the error handling activity of the task;
- x is the number of short execution times between two long execution times.

Based on this new model we again address the problem of providing a set of constraints on the execution times and activation patterns of the tasks in $\mathcal{R}$ that is sufficient to guarantee weakly-hard schedulability of all tasks τ in the nominal mode.

Another interesting model is when a task has a single execution time but with varying densities. However, this work focuses only on different execution times.

Let us first focus on a task τ_i in the nominal mode with an (m, k) constraint, i.e., τ_i may miss no more than m out of k deadlines. Denote $\mathcal{R}_i$ the set of under-specified tasks with a priority higher than τ_i, $\Omega_k^{r\to i} = \eta_r^+(\Delta T_i^k)$ for all $\tau_r \in \mathcal{R}_i$ and $\Omega_i = \sum_{r\in\mathcal{R}_i} \Omega_k^{r\to i}$.

Let us first formulate a hypothesis, which is consistent with the application scenario mentioned at the beginning of this section.

Hypothesis 1. *For each task $\tau_r \in \mathcal{R}_i$, we allow only one instance out of Ω_i^r to have a long execution time C_r^l. The other $\Omega_i^r - 1$ instances of τ_r within ΔT_i^k will be bounded by the short execution time bound C_r^s.*

In a way that is similar to TWCA as explained in Section 3.2 we now introduce the concept of combinations.

Definition 6.5. *A τ_i* combination *is a tuple $\bar{c} = (c_1, c_2, \ldots, c_{|\mathcal{R}_i|})$ such that each task $\tau_r \in \mathcal{R}_i$ corresponds to one c_r in the tuple and $c_r = 0$ or $c_r = C_r^s$ or $c_r = C_r^l$.*

We use the notation $c_r^{\bar{c}}$ to refer to the execution time of τ_r in combination $\bar{c}$. Note that we exclude here the possibility for several instances of the same under-specified task to be in the same τ_i busy-window. That is, we suppose that $\forall \tau_r \in \mathcal{R}_i : \delta_r^-(2) > BW_i^m$.

Definition 6.6. *Let $\mu(\bar{c})$ denote the maximum number of deadline-misses that may be caused by a combination $\bar{c}$. Formally we have:*

$$S_i^{\mu(\bar{c})-1} < \sum_{\tau_r \in \mathcal{R}_i} c_r^{\bar{c}} \leqslant S_i^{\mu(\bar{c})}$$

with the convention that $S_i^{-1} = 0$. If $\mu(\bar{c}) = 0$ then $\bar{c}$ is called schedulable*, otherwise it is said to be* unschedulable*.*

Of course $\mu(\bar{c})$ depends on the values chosen for the various execution times C_r^l and C_r^s for $\tau_r \in \mathcal{R}_i$. Our strategy for budgeting the under-specified tasks is to first assign values on $\mu(\bar{c})$ for all combinations and then in a second step to assign execution time budgets.

Hypothesis 2. *We suppose that a combination containing only short execution times of under-specified tasks cannot be unschedulable. That is, $\sum_{\tau_r \in \mathcal{R}_i} C_r^s \leqslant S_i^0$.*

Again this hypothesis seems realistic given the application context.

Based on the notion of combination we can define *gangs*, which correspond to distributions of the Ω_i instances within ΔT_i^k. More specifically, a gang is a packing of instances of the under-specified tasks into the τ_i busy-windows of ΔT_i^k.

Definition 6.7. *A* gang *$\mathcal{G}$ is a set of combinations that contain at least one long execution time and such that for all $\tau_r \in \mathcal{R}_i$*

- $\#\{\bar{c} \in \mathcal{G} \mid c_r^{\bar{c}} > 0\} \leqslant \Omega_k^{r \to i}$
- $\#\{\bar{c} \in \mathcal{G} \mid c_r^{\bar{c}} = C_r^l\} = 1$

Notice that we ignore combinations that do not contain any long execution time as they cannot lead to deadline-misses. Note also that each combination appears at most once in a gang (since there can be only one long execution time of each task within ΔT_i^k).

We use $\mathbb{G}_i$ to denote all possible gangs with respect to τ_i.

Lemma 6.7. *If $\forall \mathcal{G} \in \mathbb{G}_i : \sum_{\bar{c} \in \mathcal{G}} \mu(\bar{c}) \leqslant m$ then τ_i is (m,k)-schedulable.*

Proof. The above condition guarantees that no matter how instances of under-specified tasks align, they can never result in more than m deadline-misses. □

This lemma trivially extends to upper bounds on the $\mu(\bar{c})$ as we formulate now.

Lemma 6.8. *For all $\bar{c}$, let $\mu_{\bar{c}}$ be an upper bound on $\mu(\bar{c})$. If $\forall \mathcal{G} \in \mathbb{G}_i : \sum_{\bar{c} \in \mathcal{G}} \mu_{\bar{c}} \leqslant m$ then τ_i is (m,k)-schedulable.*

Now, one thing which does not appear in the above lemma is that the $\mu(\bar{c})$ are not independent from each other.

Definition 6.8. *There exists a partial order $\leqslant$ on combinations such that $\bar{c}_1 \leqslant \bar{c}_2$ if and only if the execution times in $\bar{c}_1$ are all smaller than their counterpart in $\bar{c}_2$, i.e.,*

$$\forall \tau_r \in \mathcal{R}_i : c_r^{\bar{c}_1} \leqslant c_r^{\bar{c}_2}$$

Lemma 6.9. *If $\bar{c}_1 \leqslant \bar{c}_2$ then $\mu(\bar{c}_1) \leqslant \mu(\bar{c}_2)$.*

Proof. This directly follows from the fact that $\bar{c}_1 \leqslant \bar{c}_2$ implies that the load incurred within one τ_i busy-window by the under-specified tasks in $\bar{c}_1$ is smaller than that in $\bar{c}_2$. □

Theorem 6.5. *Suppose that you have assigned the $\mu_{\bar{c}}$ such that $\forall \mathcal{G} \in \mathbb{G}_i$: $\sum_{\bar{c} \in \mathcal{G}} \mu_{\bar{c}} \leqslant m$. Then any assignment of the $c_r^{\bar{c}}$ such that for all combination $\bar{c}$, $\sum_{r \in \mathcal{R}_i} c_r^{\bar{c}} \leqslant S_i^{\mu_{\bar{c}}}$ guarantees the (m,k)-schedulability of τ_i.*

Proof. This follows directly from Lemma 6.8 and the definition of $\mu_{\bar{c}}$-slack. □

Note that there always exists such an assignment.

Now that we have presented our solution for budgeting under-specified tasks based on the multiframe execution time model, let us show how it proceeds on an illustrative example.

Example 6.1. *Consider as an example a system with only one task τ_3 in the nominal mode and two under-specified tasks τ_1 and τ_2, as illustrated in Figure 6.2. Task τ_3 has a $(2, 10)$ constraint. τ_1 and τ_2 have priorities higher than the priority of τ_3, and no more than 2 instances within ΔT_i^k.*

Figure 6.2 shows gang $\mathcal{G} = \{\bar{c}_1, \bar{c}_4, \bar{c}_7\}$ where $\bar{c}_1 = (C_1^l)$, $\bar{c}_4 = (C_1^s, C_2^l)$ and $\bar{c}_7 = (C_2^s)$ — to improve readability we omit 0s in the representation of combinations. There are five combinations containing at least one long execution time:

$$\bar{c}_1 = (C_1^l), \bar{c}_2 = (C_2^l), \bar{c}_3 = (C_1^l, C_2^s), \bar{c}_4 = (C_1^s, C_2^l), \bar{c}_5 = (C_1^l, C_2^l)$$

There are three more combinations containing at least one short execution time:

$$\bar{c}_6 = (C_1^s), \bar{c}_7 = (C_2^s), \bar{c}_8 = (C_1^s, C_2^s)$$

Let us now focus on gangs. Remember that gangs consist of combinations containing at least one long execution time and that two combinations with the long same execution time cannot be in the same gang. We only list here maximal gangs.

$$\mathcal{G}_1 = \{\bar{c}_1, \bar{c}_2\}, \mathcal{G}_2 = \{\bar{c}_1, \bar{c}_4\}, \mathcal{G}_3 = \{\bar{c}_2, \bar{c}_3\}, \mathcal{G}_4 = \{\bar{c}_3, \bar{c}_4\}, \mathcal{G}_5 = \{\bar{c}_5\}$$

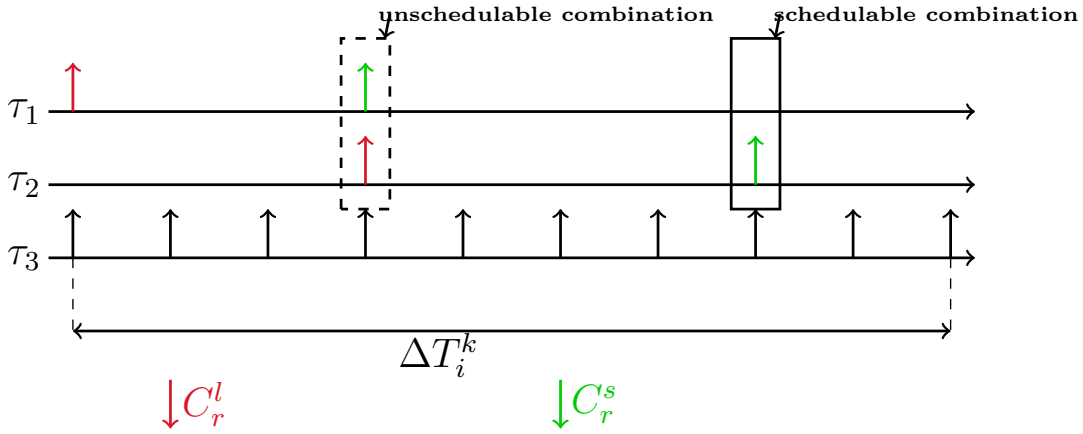

Figure 6.2: A gang of τ_1 and τ_2 within ΔT_i^k where τ_3 has a real-time constraint $(2, 10)$.

This yields the following constraints, the first five of which are directly derived from the gangs while the remaining four constraints are obtained by comparing combinations.

1. $\mu_{\bar{c}_1} + \mu_{\bar{c}_2} \leqslant 2$
2. $\mu_{\bar{c}_1} + \mu_{\bar{c}_4} \leqslant 2$
3. $\mu_{\bar{c}_2} + \mu_{\bar{c}_3} \leqslant 2$
4. $\mu_{\bar{c}_3} + \mu_{\bar{c}_4} \leqslant 2$
5. $\mu_{\bar{c}_5} \leqslant 2$
6. $\mu_{\bar{c}_1} \leqslant \mu_{\bar{c}_3}$
7. $\mu_{\bar{c}_3} \leqslant \mu_{\bar{c}_5}$
8. $\mu_{\bar{c}_2} \leqslant \mu_{\bar{c}_4}$
9. $\mu_{\bar{c}_4} \leqslant \mu_{\bar{c}_5}$

One solution to this set of constraints is, e.g., $\mu_{\bar{c}_1} = 1, \mu_{\bar{c}_2} = 1, \mu_{\bar{c}_3} = 1, \mu_{\bar{c}_4} = 1, \mu_{\bar{c}_5} = 2$.

Assuming we have chosen the above assignment for the $\mu_{\bar{c}}$*, we now have to define the constraints to be satisfied by the execution times of tasks, one per combination and then one for the short execution times.*

1. $C_1^l \leqslant S_i^1$
2. $C_2^l \leqslant S_i^1$
3. $C_1^l + C_2^s \leqslant S_i^1$
4. $C_1^s + C_2^l \leqslant S_i^1$
5. $C_1^l + C_2^l \leqslant S_i^2$
6. $C_1^s + C_2^s \leqslant S_i^0$

Any solution to this set of constraints guarantees (m, k)*-schedulability of* τ_i.

6.4 Methodology

Let us now summarize the methodology that we propose to provide the architect with *simple answers* helping him/her dimension the tasks that are still under-specified in the system.

1. We first compute an execution time budget for the under-specified tasks which guarantees *hard real-time* constraints (zero deadline-misses). If this execution time budget is acceptable for the architect then we do not need to go further.

2. If, however there is a need for larger execution times for the under-specified tasks, we then compute a second execution time budget which guarantees weakly-hard constraints. Taking into account *weakly-hard constraints* we can allow more load within shorter time windows but over longer time windows the load available for under-specified tasks is still limited.

3. If the activation patterns of the under-specified tasks are known and a *multiframe execution time model* is meaningful we can propose more relaxed bounds on execution times budgets.

6.5 Experiments

Let us now provide some experimental results we have obtained using the cplex constraint solver on budgeting under-specified tasks. We first address the motivational example of Section 6.1 and then present experiments made on synthetic test cases.

6.5.1 Industrial case study

The case study presented in Section 6.1 is a system made of a single resource and a task set shown in Table 6.1 where 27 tasks are in the nominal mode and there are 3 recovery and reconfiguration tasks τ_{10}, τ_{11}, τ_{21} which are under-specified.

As discussed before in Section 6.1, all OBSW is currently typically analyzed with hard real-time techniques; and yet by experience, the overall system is still quite robust to occasional deadline-misses, although at the moment there is no necessity to formally evaluate such tolerance in the state-of-the-practice process.

For the sake of the case study we propose some WHRT constraints for tasks that are purposely quite aggressive: the reader could notice that in some cases a tolerance of 1 deadline every 2 seconds is admitted for some tasks. This would

permit to ascertain the robustness (at least from the point of view of real-time constraints) of such representative task set even in case of severe degradation (which would require high sporadic load for the recovery activities).

The worst-case response time analysis of the nominal mode shows that the system is schedulable. Our goal is to synthesize a load budget for the under-specified tasks τ_{10}, τ_{11}, τ_{21} which guarantees that all WHRT constraints described in Table 6.2 are satisfied. We show first the constraints on the execution times and activation models of the tasks in $\mathcal{R}$ which guarantee absence of any deadline-miss before providing the same result when a few deadline-misses are tolerated.

task	τ_{12}	τ_{13}	τ_{14}	τ_{15}	τ_{16}	τ_{17}	τ_{18}	τ_{19}	τ_{20}
(m, w)	(1,2)	(1,4)	(1,8)	(1,4)	(1,4)	(1,4)	(1,8)	(1,8)	(1,8)
(m, k)	(1,16)	(1,16)	(1,8)	(1,8)	(1,16)	(1,8)	(1,8)	(1,8)	(1,8)
task	τ_{22}	τ_{23}	τ_{24}	τ_{25}	τ_{26}	τ_{27}	τ_{28}	τ_{29}	τ_{30}
(m, w)	(1,16)	hard	hard	(1,8)	(1,8)	(1,16)	(1,16)	(1,16)	hard
(m, k)	(1,8)	hard	hard	(1,8)	(1,8)	(1,8)	(1,8)	(1,8)	hard

Table 6.2: Real-time constraints of tasks in $\mathcal{N}'$. (m, w) represents the maximum number of allowed deadline-misses m every w seconds, (m, k) means that a task may miss at most m deadline out of k consecutive instances.

Note that tasks $\{\tau_1, \ldots, \tau_9\}$ have higher priority than the recovery and reconfiguration tasks so their timing properties do not depend on the budget of tasks in $\mathcal{R}$. They will therefore be excluded from our study. We denote by $\mathcal{N}'$ the remaining tasks with lower priority, that is: $\mathcal{N}' = \mathcal{N} \setminus \{\tau_1, \ldots, \tau_9\}$.

6.5.1.1 Budgeting with hard real-time constraints

If we want to guarantee that the system is schedulable then the budget to be shared between the under-specified tasks is $S_i^0 = 48.01$ms. If this budget is not sufficient for the architect we can propose a budget with WHRT guarantees.

6.5.1.2 Budgeting with weakly-hard real-time constraints

If the architect can accept to work with weakly-hard rather than hard guarantees then the available budget for the recovery tasks is $(m + 1) \times S_i^0 = 96.02$ ms.

This budget is twice as much as the budget for the hard real-time case. We can obtain even better bounds by using a more fine-grained model of how load distributes over busy-windows.

6.5.1.3 Budgeting for multiframe tasks

Let us assume that for all $\tau_i \in \mathcal{N}$ there are at most $\Omega_{10} = 1$, $\Omega_{11} = 3$ and $\Omega_{21} = 2$ instances of the under-specified tasks within ΔT_i^k. The following execution times guarantee (m, k)-schedulability of all tasks.

$$C_{11}^l = 24.005,\ C_{11}^s = 12.0025$$

$$C_{10}^l = 24.005,\ C_{10}^s = 12.0025$$

$$C_{21}^l = 24.005,\ C_{21}^s = 12.0025$$

This means in particular that the budget that is available for the under-specified tasks within ΔT_i^k is at least 108.015 ms. Note that there are many other possible assignments for the μ values which lead to different execution times.

6.5.2 Synthetic test cases

In this section, we present a set of synthetic test cases to test more extensively our approach on a variety of systems. In this experiment we study the impact of different characteristics such as utilization, (m, k) constraints, system size, etc.

For that purpose 1000 task sets were randomly generated following the approach explained in Section 4.2.1. We define a set of tasks $\mathcal{Z}$ with a priority, a worst-case execution time, a period, a deadline, and an (m, k) constraint. Remember that only the under-specified tasks have a multiframe model, therefore, we generate one worst-case execution time for the nominal tasks. We picked up a utilization among $U \in \{0.4, 0.5, 0.6, 0.7, 0.8\}$, then the number of tasks is chosen to be $\in [1, 20]$ and periods are *harmonic*. The worst-case execution time is then computed $C_i = U_i * T_i$. Deadlines $= \{0.6, 0.8, 1\} * T_i$ as our approach supports only constrained and implicit deadlines. We generate a random (m, k) for each task in the system such that: $k \in [2, 100], m \in [1, k-1]$. The number of under-specified tasks is limited to $r = 3$ and the maximum number of instances of each under-specified task is generated randomly to be in $[1, r^2]$.

6.5.2.1 Results

Figure 6.3 shows in the form of a histogram how much we gain in terms of load budget for the under-specified tasks by using a multiframe task model with weakly-hard constraints instead of using a single worst-case execution time with hard real-time constraints. Note that the results in the former case are obviously at least as good as those for the latter case.

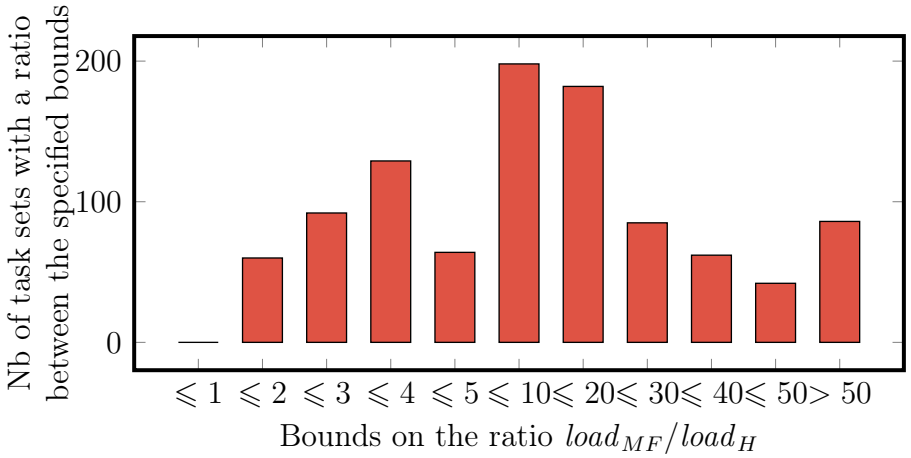

Figure 6.3: The relation between $load_{MF}$ and $load_H$ [52].

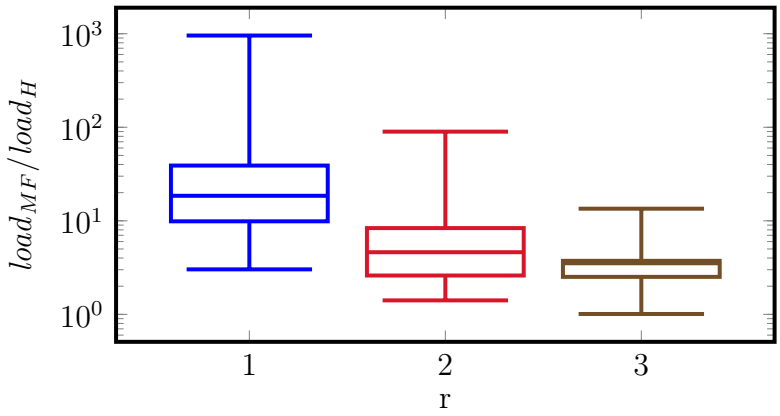

Figure 6.4: The relation between the gain of load and r [52].

Figure 6.3 shows for example that for 198 task sets the load budget in the multiframe case ($load_{MF}$) is between 5 and 10 times larger than the load budget in the hard case ($load_H$), that is:

$$5 < \frac{load_{MF}}{load_H} \leqslant 10$$

The load we gain, however, is related to the number of under-specified tasks. Figure 6.4 shows that the larger the number of under-specified tasks the less load we gain, that is due to sharing the available slack among more under-specified tasks, which makes the long execution C^l shorter.

Additional experiments show that there is no impact of the utilization on the load we gain. The number of periodic tasks causes no degradation on the load we

gain by using multiframe task model. Note that we have repeated our experiment 10 times, i.e., we generated 10 groups of task sets each of which includes 1000 task sets, and observed similar results.

6.6 Related Work

The work presented in this chapter most closely relates to sensitivity analysis, slack analysis, multiframe task systems and WHRT systems. Note that determination of bounds on unspecified system parameters is the scope of *Parametric Model Checking* [29] [57]. Even if such approaches are known to have difficulty scaling up to even simple settings, it would be interesting to see if these approaches could apply to our problem.

This work focuses on budgeting under-specified tasks for WHRT systems. Although the under-specified tasks in our case study (OBSW) are recovery tasks, schedulability analysis of fault-tolerant real-time systems [15] [68] is not in the scope of this work.

Sensitivity analysis is used to provide guarantees on the schedulability of a system in case of uncertainty on the system parameters. In [10] Bini et al. introduced an analytical sensitivity analysis for FPP scheduled periodic task sets with constrained deadlines (i.e., $D \leqslant T$). Work by [123] and [81] propose solutions for sensitivity analysis of systems with activation patterns specified with arrival curves. Racu et al. [93] applied sensitivity analysis in DRTSs.

In contrast to all these papers, our work proposes for the first time a solution for the sensitivity analysis of WHRT systems: We constrain the admissible load that under-specified tasks in the system can use without violating WHRT constraints in FPP scheduled task sets with arbitrary activation patterns and constrained deadlines.

Slack stealing is a scheduling algorithm proposed by [66] to schedule aperiodic tasks by stealing all the processing time it can from the periodic tasks without causing their deadlines to be missed. Similar algorithms based on slack stealing have been proposed by other authors [28] [98] [117]. These algorithms do not take into account any weakly hard guarantees and they, therefore, bound the maximum slack in a window of size D_i. In our approach, however, we consider (m, k) weakly-hard requirements and we thus bound the maximum slack in a window of size $\delta_i^-(m+1) + D_i$.

The multiframe task model was proposed originally in [77] to provide a less pessimistic schedulability test than [69] for hard real-time systems. This model assigns to each *periodic* task N execution times $(C^0, C^1, \ldots, C^N)$, the execution time alternates between them where the execution time of the i-th instance of

the task is $C^{((i-1)modN)}$ where $i \geqslant 1$. In this work we use a specific case of the multiframe task model for *sporadic* tasks that assigns two execution times: long C^l and short C^s where within a time window Δ one instance of the task uses the long execution time while the rest use the short execution times. We propose our multiframe task model in a context of budgeting *under-specified* recovery tasks (sporadic) to provide the recovery tasks with more load for WHRT systems.

Weakly-hard systems [7] is a concept that guarantees that out of k consecutive executions of a task, not more than m deadline-misses may occur. The approach of [91] and the related articles provide analyses to verify such constraints. In this work we reuse the concepts developed in these papers to better budget under-specified tasks.

6.7 Summary

In this chapter, we have shown how to budget under-specified tasks in the early design of WHRT systems by providing sufficient conditions, which guarantee (m, k) schedulability. This is particularly useful in industrial practice because it often happens during design that some parts of a task set are fully specified while other parameters, e.g., regarding recovery or monitoring tasks, do not become available before much later. Existing budgeting techniques, which are restricted to hard real-time constraints, can help anticipate how these missing parameters influence the behavior of the whole system, but they are likely to yield execution time budgets that are too tight to be useful. We have shown that using weakly-hard rather than hard guarantees, whenever possible, results in much more applicable execution time budgets. The results are thus of real practical value for the design of systems such as the on-board software system discussed in the chapter.

Note that in this chapter we have not at all addressed the issue of the complexity of the analysis. The reason for that is that this does not appear to be a limiting factor for industrial applicability at this point. It would however be interesting to better understand how far the presented approach can scale and how much we can improve its efficiency.

7 | CONCLUSION

The pairing of the cyber and physical in *cyber-physical systems* (CPS) [95] imposes different levels of safety requirements depending on how critical the functions assigned to the system are and on how humans interact with the system. Such safety requirements involve timing constraints, the violation of which may lead to a system failure. Hence, not only applications of classical domain, e.g., automotive, avionics, etc., but also applications of emerging technology may have timing requirements. Timing constraints are graded from soft to hard real-time constraints. While satisfying soft real-time constraints requires only best-effort guarantees, hard real-time constraints are best treated with worst-case analysis methods for verifying all timing constraints. WHRT systems have extra demands on the timing verification as they tolerate few deadline-misses in certain distributions. In worst-case analysis methods, a task is schedulable only when it can meet its deadline in the worst-case, therefore, guarantees computed using worst-case analysis methods are not satisfactory for WHRT systems. Considering tolerable deadline-misses raises the need for weakly-hard schedulability analyses to verify WHRT constraints ((m, k) constraints) and to provide more expressive guarantees.

This thesis addressed the schedulability analysis problem of WHRT systems. It presented an efficient analysis to compute WHRT guarantees in the form of a deadline miss model for various system models. The core contribution of this thesis was a deadline miss model for a temporarily overloaded uniprocessor system with independent tasks under the FPP and FPNP scheduling policies using TWCA. In our application context, the transient overload is due to sporadic tasks, for example interrupt service routines. The proposed analysis was adopted to compute deadline miss models for independent tasks under the WRR and EDF scheduling policies.

The thesis exposed the complexity of computing a DMM and it showed the necessity for a compromise between the pessimism of the computed guarantees and the time efficiency of the analysis. Experimental results showed the scalability of TWCA w.r.t. k and the system size. Interestingly, the results illustrated how the scheduling policy alters the impact that sporadic overload has on the quality of the results. The EDF scheduling policy provided the best DMMs but at the expense of missing the predictability, i.e., every task is susceptible to deadline-miss, and the

possibility of suffering from the domino effect. FPNP, however, provided better DMMs comparing to FPP and RM and at the same time it kept the predictability due to its fixed priority behavior.

A part of this work has been achieved in the course of a collaboration project between Thales Research & Technology and Technische Universität Braunschweig. The collaboration project involved two case studies, which necessitated extensions to the core contribution in order to tackle the problem raised by these case studies. The first extension presented an analysis to compute deadline miss models for WHRT systems with task chains under FPP. The extension showed how TWCA can be extended using recent results in the analysis of hard real-time systems with task dependencies. The approach was validated on the first case study, which was inspired by industrial practice, and synthetic variants of it. The second extension is dedicated to budget recovery and reconfiguration tasks in a WHRT system such that tasks of the system guarantee (m, k) schedulability. This extension handled the second case study. Existing budgeting techniques, which are restricted to hard real-time constraints, can help anticipate how these missing parameters influence the behavior of the whole system, but they are likely to yield execution time budgets that are too tight to be useful. The proposed analysis showed that using weakly-hard rather than hard guarantees, whenever possible, results in much more applicable execution time budgets.

Although this thesis touched upon an already addressed topic, i.e., WHRT systems, the analysis presented in this thesis is of high flexibility allowing us to consider various scheduling policies (FPP, FPNP, EDF, WRR) and to cover both independent and dependent tasks. The analysis is of manageable computation cost and scales well with k and the system size. Furthermore, the proposed analysis is compatible with CPA which makes it extendible to DTRSs which permits to provide end-to-end deadline miss models for more complex task models.

The thesis provided two practical solutions for two industrial case studies, which are involved exclusively in the collaboration project. The results are thus of real practical value to be considered in the design process of WHRT systems.

7.1 Applications of TWCA Out of This Thesis

TWCA has been applied to case studies outside of this thesis. The following list collects and summarizes the works that exploited TWCA:

• **CAN bus analysis with Daimler.** [89] demonstrates how TWCA can be used to analyze a real CAN bus with complex activation patterns. Authors investigated the effects of these load patterns and showed how the necessary parameters can be derived and verified from traces and specifications.

• **Case studies with Bosch.** It has been shown also in [39] that the weakly-hard guarantees provided by TWCA can be used, e.g., for efficient verification of closed-loop properties of control software.

• **TWCA at runnable granularity.** A variant of TWCA is provided in [1] that is fine-grained enough to provide hard and weakly-hard response time guarantees for runnable entities. If real-time guarantees are only available at task granularity, the strictest real-time requirement of a runnable entity determines the real-time requirement of the entire task. However, by giving real-time guarantees for each runnable entity, this over-provisioning can be avoided.

• **Typical worst-case execution time.** In [119], authors extended TWCA to cope with periodic tasks that have varying execution times. Taking the robustness of control applications into account, they derived upper bounds for the overload models of each task, along with possible typical worst-case execution times (TCET), as needed for the TWCA.

• **Anomaly detection.** The timing behavior of a safety-critical system could be considered as a prediction mechanism of various attacks against such system. In [46], authors proposed a prediction configuration for real-time tasks with temporal constraints to predict the abnormal behavior of these tasks. They used in particular the TWCRT to define a bound that represents the least privilege temporal bound for each real-time task.

7.2 Ongoing Work and Outlook

We have shown throughout this thesis that TWCA is an extendible framework that can be generalized to cover more more complex system models such as DRTSs. In fact, there is a running project considering the contribution of this thesis to provide WHRT guarantees in the form of a deadline miss model for a DRTS. The project, which is called TypicalCPA, is funded by the Deutsche Forscungsgemeinschaft[1] (DFG) [37]. Ahrendts et al. in [2] presented a CPA extension of TWCA to compute an end-to-end WHRT guarantees for switched network. However, the presented DMM in this thesis is over-approximated by two main sources of pessimism, which have been formally and experimentally determined in Chapter 4. There is therefore room for providing a more tight DMM with feasible complexity by tackling these two sources of pessimism.

The state-of-the-art in the analysis of safety-critical real-time systems, including this thesis, is based on pen-and-paper proofs, which are hard to reuse and may

[1]The project is under the contract number 168/30-2.

contain errors. A way to avoid this is to require that timing analysis results be formally proven, machine-checkable, and independently verifiable. To this end, the RT-PROOFS project [86] is laying the foundations for the computer-assisted verification of schedulability analysis results by (i) formalizing foundational real-time concepts using the Coq proof assistant and (ii) mechanizing proofs of busy-window-based end-to-end latency analysis, the analysis approach of greatest practical relevance (e.g., used by SymTA/S). In this context, Fradet et al. presented in [38] a formal proof of TWCA based on this thesis using the Coq proof assistant [87].

Chapter 6 proposed a formal analysis to check the capability of scheduling a new task in a WHRT system. That assists in establishing an accepting/rejecting process for dynamic systems. The project controlling concurrent change (CCC) [22], which is funded by DFG, aims to use formal models and methods to do *in-field* integration and test. CCC addresses critical applications where updates and new functions can be integrated in the field. In-field integration imposes therefore appropriate methods and platform architectures ensuring the same quality as lab based integration. [78].

Finally, we need to acknowledge the need for complementary work related to WHRT systems as mentioned in Section 6.1, in particular in relation with the impact of deadline-misses on system functions. Recent work [39, 12, 85] in this direction indicate that this question is indeed considered as relevant in the research community as well as in the industry.

PUBLICATIONS

Related to this thesis

- Zain A. H. Hammadeh and Rolf Ernst. Weakly-hard real-time guarantees for weighted round-robin scheduling of real-time messages. In IEEE 23rd International Conference on Emerging Technologies and Factory Automation, Turin, Italy, September 2018.

- Zain A. H. Hammadeh, Sophie Quinton, and Rolf Ernst, "Weakly-Hard Real-Time Guarantees for Earliest Deadline First Scheduling of Independent Real-Time Tasks" in ACM Transactions on Embedded Computing Systems, 2018. Under review.

- Zain A. H. Hammadeh, Sophie Quinton, Marco Panunzio, Rafik Henia, Laurent Rioux, and Rolf Ernst, "Budgeting Under-specified Tasks for Weakly-Hard Real-Time Systems" in The 29th Euromicro Conference on Real-Time Systems (ECRTS17), (Dubrovnik, Croatia), June 2017.

- Rafik Henia, Laurent Rioux, Nicolas Sordon, Zain A. H. Hammadeh, Sophie Quinton, and Rolf Ernst, "Demo Abstract: Bounding Deadline Misses for Weakly-Hard Real-Time Systems Designed in CAPELLA" in The 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) - Demo Track, (Pittsburgh, PA, USA), April 2017.

- Zain A. H. Hammadeh, Sophie Quinton, Rafik Henia, Laurent Rioux, and Rolf Ernst, "Bounding Deadline Misses in Weakly-Hard Real-Time Systems with Task Dependencies" in Design Automation and Test in Europe (DATE), (Lausanne, Switzerland), March 2017.

- Wenbo Xu, Zain A. H. Hammadeh, Sophie Quinton, Alexander Kröller, and Rolf Ernst, "Improved Deadline Miss Models for Real-Time Systems using Typical Worst-Case Analysis" in 27th Euromicro Conference on Real-Time Systems, (Lund, Sweden), July 2015.

- Zain A. H. Hammadeh, Sophie Quinton, and Rolf Ernst, "Extending Typical Worst-Case Analysis Using Response-Time Dependencies to Bound Deadline Misses" in Proceedings of the 14th International Conference on Embedded Software, (New Delhi, India), October 2014

Not related to this thesis

- Mohammad Hamad, Zain A. H. Hammadeh, Selma Saidi, Vassilis Prevelakis und Rolf Ernst, "Prediction of Abnormal Temporal Behavior in Real-Time Systems" in The 33rd ACM/SIGAPP Symposium On Applied Computing (SAC 2018), (Pau, France), April 2018.

- Leonie Ahrendts, Zain A. H. Hammadeh, and Rolf Ernst, "Guarantees for Runnable Entities with Heterogeneous Real-Time Requirements" in Design, Automation & Test in Europe Conference & Exhibition (DATE), (Dresden, Germany), March 2016.

List of Figures

List of Tables

List of Algorithms

Abbreviations

ATM Asynchronous Transfer Mode.

CAN Controller Area Network.

CCC Controlling Concurrent Change.

CDR Critical Design Review.

CFG Control Flow Graph.

CPA Compositional Performance Analysis.

CPS Cyber-Physical System.

DFG Deutsche Forscungsgemeinschaft.

DM Deadline Monotonic.

DMM Deadline Miss Model.

DRTS Distributed Real-Time System.

ECU Electronic Control Unite.

EDF Earliest Deadline First.

FCFS First-Come-First-Serve.

FIFO First-In-First-Out.

FPGA Field-Programmable Gate Array.

FPNP Fixed Priority NonPreemptive.

FPP Fixed Priority Preemptive.

GPP General Purpose Processor.

GPU Graphics Processing Unit.

ILP Integer Linear Programming.

LP Linear Programming.

MILP Mixed-Integer Linear Programming.

NP Nondeterministic Polynomial time.

OBSW On-Board SoftWare.

QoS Quality of Service.

QPA Quick convergence Processor-demand Analysis.

RM Rate-Monotonic.

RTC Real-Time Calculus.

TAS Thales Alenia Space.

TCET Typical worst-Case Execution Time.

TDMA Time-Devision Muliple Access.

TRT Thales Research & Technology.

TWCA Typical Worst-Case Analysis.

TWCRT Typical Worst-Case Response Time.

WCL Worst-Case Latency.

WHRT Weakly-Hard Real-Time.

WRR Wighted Round-Robin.

Bibliography

[1] Leonie Ahrendts, Zain A. H. Hammadeh, and Rolf Ernst. Guarantees for runnable entities with heterogeneous real-time requirements. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016.

[2] Leonie Ahrendts, Sophie Quinton, Thomas Boroske, and Rolf Ernst. Verifying Weakly-Hard Real-Time Properties of Traffic Streams in Switched Networks. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:22, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[3] Leonie Ahrendts, Sophie Quinton, and Rolf Ernst. Finite ready queues as a mean for overload reduction in weakly-hard real-time systems. In *RTNS*, Grenoble, France, 2017.

[4] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, Jan 2004.

[5] Philip Axer. *Performance of Time-Critical Embedded Systems under the Influence of Errors and Error Handling Protocols*. PhD thesis, TU Braunschweig, 2015.

[6] Philip Axer, Maurice Sebastian, and Rolf Ernst. Probabilistic response time bound for can messages with arbitrary deadlines. In *Proceedings of Design, Automation and Test in Europe*, Dresden, Germany, 2012.

[7] Guillem Bernat, Alan Burns, and Albert Llamosí. Weakly hard real-time systems. *IEEE Trans. Computers*, 50(4):308–321, 2001.

[8] Guillem Bernat and Ricardo Cayssials. Guaranteed on-line weakly-hard real-time systems. In *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (Cat. No.01PR1420)*, pages 25–35, Dec 2001.

[9] Enrico Bini and GiorgioC. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[10] Enrico Bini, Marco Di Natale, and Giorgio C. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. In *18th Euromicro Conference on Real-Time Systems, ECRTS'06, 5-7 July 2006, Dresden, Germany, Proceedings*, pages 13–22, 2006.

[11] Sara R. Biyabani, John A. Stankovic, and Krithi Ramamritham. The integration of deadline and criticalness in hard real-time scheduling. In *Proceedings. Real-Time Systems Symposium*, pages 152–160, Dec 1988.

[12] Rainer Blind and Frank Allgöwer. Towards networked control systems with guaranteed stability: Using weakly hard real-time constraints to model the loss process. In *54th IEEE Conference on Decision and Control, CDC 2015, Osaka, Japan, December 15-18, 2015*, pages 7510–7515, 2015.

[13] Markus Bohlin, Yue Lu, Johan Kraft, Per Kreuger, and Thomas Nolte. Simulation-based timing analysis of complex real-time systems. In *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 321–328, Aug 2009.

[14] Torsten Bone. Applying timing analysis to vehicle networking at daimler group research and advanced engineering, September 2010.

[15] Alan Burns, Robert Davis, and Sasikumar Punnekkat. Feasibility analysis of fault-tolerant real-time task sets. In *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems*, pages 29–33, Jun 1996.

[16] Niko A. Busch and Rufin VanRullen. Spontaneous eeg oscillations reveal periodic sampling of visual attention. *Proceedings of the National Academy of Sciences*, 107(37):16048–16053, 2010.

[17] Giorgio C. Buttazzo. Rate monotonic vs. edf: Judgment day. *Real-Time Syst.*, 29(1):5–26, January 2005.

[18] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer Publishing Company, Incorporated, 3rd edition, 2011.

[19] Giorgio C. Buttazzo and John A. Stankovic. Red: Robust earliest deadline scheduling. In *PROC. OF 3RD INTERNATIONAL WORKSHOP ON RESPONSIVE COMPUTING SYSTEMS*, pages 100–111, 1993.

[20] Laura Carnevali, Alessandra Melani, Luca Santinelli, and Giuseppe Lipari. Probabilistic deadline miss analysis of real-time systems using regenerative transient analysis. In *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, RTNS '14, pages 299:299–299:308, New York, NY, USA, 2014. ACM.

[21] Francisco J. Cazorla, Eduardo Quiñones, Tullio Vardanega, Liliana Cucu, Benoit Triquet, Guillem Bernat, Emery D. Berger, Jaume Abella, Franck Wartel, Michael Houston, Luca Santinelli, Leonidas Kosmidis, Code Lo, and Dorin Maxim. PROARTIS: Probabilistically analyzable real-time systems. *ACM Trans. Embedded Comput. Syst.*, 12(2s):94, 2013.

[22] Controlling Concurrent Change (CCC). URL: `http://ccc-project.org//`.

[23] Anton Cervin. Analyzing effects of missed deadlines in control systems. In *ARTES Graduate Student Conference*, 2001.

[24] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proceedings of DATE'03*, pages 190–195. IEEE Computer Society, 2003.

[25] Kuan-Hsun Chen and Jian-Jia Chen. Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors. In *2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–8, June 2017.

[26] Robert I. Davis. On the evaluation of schedulability tests for real-time scheduling algorithms. In *Proceedings international workshop on analysis tools and methodologies for embedded and real-time systems (WATERS)*, 2016.

[27] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[28] Robert I. Davis, Ken W. Tindell, and Alan Burns. Scheduling slack time in fixed priority pre-emptive systems. In *Real-Time Systems Symposium, 1993., Proceedings.*, pages 222–231, Dec 1993.

[29] Conrado Daws. Symbolic and parametric model checking of discrete-time markov chains. In *Proceedings of the First International Conference on Theoretical Aspects of Computing*, ICTAC'04, pages 280–294. Springer-Verlag, 2005.

[30] Michael L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.

[31] José L. Díaz, Daniel F. García, Kanghee Kim, Chang-Gun Lee, Lucia Lo Bello, José M. López, Sang Lyul Min, and Orazio Mirabella. Stochastic analysis of periodic real-time systems. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 289–300, 2002.

[32] Jonas Diemer. *Predictable Architecture and Performance Analysis for General-Purpose Networks-on-Chip*. PhD thesis, TU Braunschweig, 2016.

[33] Jonas Diemer, Philip Axer, and Rolf Ernst. Compositional performance analysis in python with pycpa. In *3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, Jul 2012.

[34] ECSS Standard E-ST-40C. Software general requirements. URL: `http://ecss.nl/standard/ecss-e-st-40c-software-general-requirements/`.

[35] Rolf Ernst. Beyond the deadline: New interfaces between control and scheduling for the design and analysis of critical embedded systems, 2017. ESWEEK - Tutorial Slides.

[36] Rolf Ernst. Rechnerstrukturen ii, 2017. TU Braunschweig, URL: `https://www.ida.ing.tu-bs.de/lehre/veranstaltungen/ag-ernst/rs2/`.

[37] Deutsche Forschungsgemeinschaft (German Research Foundation). URL: `http://www.dfg.de/en/`.

[38] Pascal Fradet, Maxime Lesourd, Jean-François Monin, and Sophie Quinton. A Generic Coq Proof of Typical Worst-Case Analysis. In *RTSS 2018 - 39th IEEE Real-Time Systems Symposium*, pages 1–12, Nashville, United States, December 2018.

[39] Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*, pages 53–62, 2014.

[40] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[41] W. Geelen, Duarte Antunes, Jeroen P. M. Voeten, Ramon R. H. Schiffelers, and W. P. M. H. Heemels. The impact of deadline misses on the control

performance of high-end motion control systems. *IEEE Transactions on Industrial Electronics*, 63(2):1218–1229, Feb 2016.

[42] Alain Girault, Christophe Prévot, Sophie Quinton, Rafik Henia, and Nicolas Sordon. Improving and estimating the precision of bounds on the worst-case latency of task chains. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2578–2589, Nov 2018.

[43] N. Guan and W. Yi. General and efficient response time analysis for edf scheduling. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1–6, March 2014.

[44] José C. Palencia Gutiérrez and Michael González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of RTSS'19*, pages 26–37, 1998.

[45] José C. Palencia Gutiérrez and Michael González Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *Proceedings of RTSS'99*, pages 328–339, 1999.

[46] Mohammad Hamad, Zain A. H. Hammadeh, Selma Saidi, Vassilis Prevelakis, and Rolf Ernst. Prediction of abnormal temporal behavior in real-time systems. In *The 33rd ACM/SIGAPP Symposium On Applied Computing (SAC 2018)*, 2018.

[47] Moncef Hamdaoui and Parameswaran Ramanathan. A dynamic priority assignement technique for streams with (m, k)-firm deadlines. *IEEE Trans. Computers*, 44(12):1443–1451, 1995.

[48] Zain A. H. Hammadeh and Rolf Ernst. Weakly-hard real-time guarantees for weighted round-robin scheduling of real-time messages. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 384–391, Sept 2018.

[49] Zain A. H. Hammadeh, Rolf Ernst, Sophie Quinton, Rafik Henia, and Lourent Rioux. Bounding deadline misses in weakly-hard real-time systems with task dependencies. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 584–589, March 2017.

[50] Zain A. H. Hammadeh, Sophie Quinton, and Rolf Ernst. Extending typical worst-case analysis using response-time dependencies to bound deadline misses. In *Proceedings of the 14th International Conference on Embedded Software*, EMSOFT '14, pages 10:1–10:10. ACM, 2014.

[51] Zain A. H. Hammadeh, Sophie Quinton, and Rolf Ernst. Weakly-hard real-time guarantees for earliest deadline first scheduling of independent real-time tasks. *ACM Transactions on Embedded Computing Systems*, 2018. Under review.

[52] Zain A. H. Hammadeh, Sophie Quinton, Marco Panunzio, Rafik Henia, Laurent Rioux, and Rolf Ernst. Budgeting under-specified tasks for weakly-hard real-time systems. In *The 29th Euromicro Conference on Real-Time Systems (ECRTS17)*, Dubrovnik, Croatia, June 2017.

[53] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis — the SymTA/S approach. In *IEEE Proceedings Computers and Digital Techniques*, 2005.

[54] Rafik Henia and Laurent Rioux. Industrial verification challenge 2015, 2015. WATERS'15, URL: `http://waters2015.inria.fr/challenge/`.

[55] Rafik Henia, Laurent Rioux, Nicolas Sordon, Zain A. H. Hammadeh, Sophie Quinton, and Rolf Ernst. Demo abstract: Bounding deadline misses for weakly-hard real-time systems designed in capella. In *The 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) - Demo Track*, Pittsburgh, PA, USA, April 2017.

[56] Kai Hu, Teng Zhang, Zhibin Yang, and Wei-Tek Tsai. Simulation of real-time systems with clock calculus. *Simulation Modelling Practice and Theory*, 51:69 – 86, 2015.

[57] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits Vaandrager. Linear parametric model checking of timed automata. *The Journal of Logic and Algebraic Programming*, 52:183 – 220, 2002.

[58] Matthias Ivers and Rolf Ernst. Probabilistic network loads with dependencies and the effect on queue sojourn times. In *Proceedings of QSHINE'09*, volume 22 of *LNCS*, pages 280–296. Springer, 2009.

[59] Marek Jersák. *Compositional Performance Analysis for Complex Embedded Applications*. PhD thesis, TU Braunschweig, 2005.

[60] Leonie Köhler and Rolf Ernst. Improving a compositional timing analysis framework for weakly-hard real-time systems. In *The 25th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2019)*, Montreal, Canada, 2019.

[61] Gilad Koren and Dennis Shasha. Dover: an optimal on-line scheduling algorithm for overloaded real-time systems. In *[1992] Proceedings Real-Time Systems Symposium*, pages 290–299, Dec 1992.

[62] Pratyush Kumar and Lothar Thiele. Quantifying the effect of rare timing events with settling-time and overshoot. In *Proceedings of RTSS'33*, pages 149–160, 2012.

[63] Phillip A. Laplante. *Real-Time Systems Design and Analysis*. Wiley-IEEE Press, 3rd edition, 2004.

[64] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: A theory of deterministic queuing systems for the Internet*, volume 2050 of LNCS. Springer, 2001.

[65] John P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–213, 1990.

[66] John P. Lehoczky and Sandra Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Real-Time Systems Symposium, 1992*, pages 110–123, Dec 1992.

[67] Jian Li, YeQiong Song, and F. Simonot-Lion. Schedulability analysis for systems under (m,k)-firm constraints. In *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.*, pages 23–30, Sept 2004.

[68] George Lima and Alan Burns. Scheduling fixed-priority hard real-time tasks in the presence of faults. In *Proceedings of the Second Latin-American Conference on Dependable Computing*, LADC'05, pages 154–173, 2005.

[69] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[70] Carey Douglass Locke. *Best-effort Decision-making for Real-time Scheduling*. PhD thesis, Carnegie Mellon University, 1986.

[71] José María López, José Luis Díaz, Joaquín Entrialgo, and Daniel F. García. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-Time Systems*, 40(2):180–207, 2008.

[72] Luxoft. Model-based timing analysis & optimization. URL: `https://auto.luxoft.com/uth/timing-analysis-tools/`.

[73] António P. Magalhães, Mário Z. Rela, and João G. Silva. Deadlines in real-time systems. Technical report, Universidade do Porto, Portugal, 1993.

[74] Michael J. Magazine and Maw-Sheng Chern. A note on approximation schemes for multidimensional knapsack problems. *Mathematics of Operations Research*, 9(2):244–247, 1984.

[75] Ma Maode and H. Babak. A fault-tolerant strategy for real-time task scheduling on multiprocessor system. In *Parallel Architectures, Algorithms, and Networks, 1996. Proceedings., Second International Symposium on*, pages 544–546, Jun 1996.

[76] MathWorks. Simulink - simulation and model-based design. URL: `https://www.mathworks.com/products/simulink`.

[77] Aloysius K. Mok and Deji Chen. A multiframe model for real-time tasks. *IEEE Trans. Software Eng.*, 23(10):635–645, 1997.

[78] Mischa Möstl, Johannes Schlatow, Rolf Ernst, Nikil Dutt, Ahmed Nassar, Amir Rahmani, Fadi J. Kurdahi, T.homas Wild, Armin Sadighi, and Andreas Herkersdorf. Platform-centric self-awareness as a key enabler for controlling changes in cps. *Proceedings of the IEEE*, 106(9):1543–1567, Sept 2018.

[79] Yannick Moy, Emmanuel Ledinot, Hervé Delseny, Virginie Wiels, and Benjamin Monate. Testing or formal verification: Do-178c alternatives and industrial experience. *IEEE Software*, 30(3):50–57, May 2013.

[80] Marco Di Natale. Beyond the m-k model: restoring performance considerations in the time abstraction, 2017. ESWEEK - Tutorial Slides.

[81] Moritz Neukirchner, Sophie Quinton, Tobias Michaels, Philip Axer, and Rolf Ernst. Sensitivity analysis for arbitrary activation patterns in real-time systems. In *Proc. of Design Automation and Test in Europe (DATE)*, March 2013.

[82] Moritz Neukrichner. *Establishing Sufficient Temporal Independent Efficiently*. PhD thesis, TU Braunschweig, 2014.

[83] Linwei Niu and Gang Quan. System wide dynamic power management for weakly hard real-time systems. *J. Low Power Electronics*, 2(3):342–355, 2006.

[84] Paolo Pazzaglia, Marco Di Natale, Giorgio Buttazzo, and Matteo Secchiari. A framework for the co-simulation of engine controls and task scheduling. In Antonio Cerone and Marco Roveri, editors, *Software Engineering and Formal Methods*, pages 438–452, Cham, 2018. Springer International Publishing.

[85] Paolo Pazzaglia, Luigi Pannocchi, Alessandro Biondi, and Marco Di Natale. Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:22, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[86] RT-Proofs Project. URL: `http://rt-proofs.inria.fr/`.

[87] The Coq proof assistant. URL: `https://coq.inria.fr/`.

[88] Gang Quan and Xiaobo Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. In *Proceedings 21st IEEE Real-Time Systems Symposium*, pages 79–88, 2000.

[89] Sophie Quinton, Torsten T. Bone, Julien Hennig, Moritz Neukirchner, Mircea Negrean, and Rolf Ernst. Typical worst case response-time analysis and its use in automotive network design. In *Proceedings of DAC*, pages 1–6. ACM, 2014.

[90] Sophie Quinton, Rolf Ernst, Dominique Bertrand, and Patrick Meumeu Yomsi. Challenges and new trends in probabilistic timing analysis. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 810–815, March 2012.

[91] Sophie Quinton, Matthias Hanke, and Rolf Ernst. Formal analysis of sporadic overload in real-time systems. In *Proceedings of DATE'12*, pages 515–520. IEEE, 2012.

[92] Sophie Quinton, Mircea Negrean, and Rolf Ernst. Formal analysis of sporadic bursts in real-time systems. In *Proceedings of DATE'13*, pages 767–772, 2013.

[93] Razvan Racu, Marek Jersak, and Rolf Ernst. Applying sensitivity analysis in real-time distributed systems. In *11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Francisco, USA, Mar 2005.

[94] Amitava Raha, Nicholas Malcolm, and Wei Zhao. Hard real-time communications with weighted round robin service in atm local area networks. In

Engineering of Complex Computer Systems, 1995. Held jointly with 5th CSE-SAW, 3rd IEEE RTAW and 20th IFAC/IFIP WRTP, Proceedings., First IEEE International Conference on, pages 96–103, Nov 1995.

[95] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: The next computing revolution. In *Design Automation Conference*, pages 731–736, June 2010.

[96] Krithivasan Ramamritham and John A. Stankovic. Dynamic task scheduling in hard real-time distributed systems. *IEEE Software*, 1(3):65–75, July 1984.

[97] Parameswaran Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):549–559, Jun 1999.

[98] Sandra Ramos-Thuel and John P. Lehoczky. On-line scheduling of hard deadline aperiodic tasks in fixed-priority systems. In *Real-Time Systems Symposium, 1993., Proceedings.*, pages 160–171, Dec 1993.

[99] Kai Richter. *Compositional Scheduling Analysis Using Standard Event Models*. PhD thesis, TU Braunschweig, 2005.

[100] Kai Richter, Marek Jersák, and Rolf Ernst. A formal approach to mpsoc performance verification. *Computer*, 36(4):60–67, April 2003.

[101] RiskAMP. Whitepaper: What is monte carlo simulation? URL: `https://www.riskamp.com/library`.

[102] Paul Rubel, Mathew Gillen, Joseph Loyall, Richard Schantz, Aniruddha Gokhale, Jaiganesh Balasubramanian, Aaron Paulos, and Priya Narasimhan. Fault tolerant approaches for distributed real-time and embedded systems. In *MILCOM 2007 - IEEE Military Communications Conference*, pages 1–8, Oct 2007.

[103] Luca Santinelli and Liliana Cucu-Grosjean. A probabilistic calculus for probabilistic real-time systems. *ACM Trans. Embed. Comput. Syst.*, 14(3):52:1–52:30, April 2015.

[104] Oliver Scheickl. *Timing Constraints in Distributed Development of Automotive Real-time Systems*. PhD thesis, TU München, 2011.

[105] Johannes Schlatow and Rolf Ernst. Response-time analysis for task chains in communicating threads. In *Proceedings of RTAS'16*, 2016.

[106] Johannes Schlatow and Rolf Ernst. Response-time analysis for task chains with complex precedence and blocking relations. *ACM Trans. Embed. Comput. Syst.*, 16(5s):172:1–172:19, September 2017.

[107] Simon Schliecker. *Performance Analysis of Multiprocessor Real-Time Systems with Shared Resources.* PhD thesis, TU Braunschweig, 2011.

[108] Kang Shin, C. Krishna, and Yann-Hang Lee. A unified method for evaluating real-time computer controllers and its application. *IEEE Transactions on Automatic Control*, 30(4):357–366, Apr 1985.

[109] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts.* Wiley Publishing, 8th edition, 2008.

[110] Marco Spuri. Analysis of deadline schedule real-time systems. Technical report, INRIA, France, 1996.

[111] John A. Stankovic, Krithi Ramamritham, and Marco Spuri. *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms.* Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[112] John A. Stankovic and Krithivasan Ramamritham, editors. *Tutorial: Hard Real-time Systems.* IEEE Computer Society Press, Los Alamitos, CA, USA, 1989.

[113] Youcheng Sun and Marco Di Natale. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. *International Conference on Embedded Software (EMSOFT), ACM Transactions on Embedded Computing Systems ESWEEK Special Issue*, Oct 2017.

[114] Thales Alenia Space (TAS). URL: `https://www.thalesgroup.com/en/space`.

[115] Daniel Thiele, Jonas Diemer, Philip Axer, Rolf Ernst, and Jan Seyler. Improved formal worst-case timing analysis of weighted round robin scheduling for ethernet. In *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, Sept 2013.

[116] Lothar Thiele, Samarjit Chakraborty, Matthias Gries, Alexander Maxiaguine, and Jonas Greutert. Embedded software in network processors - models and algorithms. In *Proceedings of the First International Workshop on Embedded Software*, EMSOFT '01, pages 416–434, London, UK, UK, 2001. Springer-Verlag.

[117] Too-Seng Tia, Jane W. S. Liu, and Mallikarjun Shankar. Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems. *Real-Time Systems*, 10(1):23–43, 1996.

[118] Ken Tindell, Alan Burns, and Andy J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

[119] Sebastian Tobuschat, Rolf Ernst, Arne Hamann, and Dirk Ziegenbein. System-level timing feasibility test for cyber-physical automotive systems. In *11th IEEE International Symposium on Industrial Embedded Systems*, May 2016.

[120] Georg von der Brüggen, K. H. Chen, W. H. Huang, and J. J. Chen. Systems with dynamic real-time guarantees in uncertain and faulty execution environments. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 303–314, Nov 2016.

[121] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, and Katharina Morik. Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:22, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[122] Ernesto Wandeler. *Modular performance analysis and interface based design for embedded real time systems*. PhD thesis, ETH Zurich, 2006.

[123] Ernesto Wandeler and Lothar Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *Proceedings of the 5th ACM International Conference on Embedded Software*, EMSOFT '05, pages 80–89, 2005.

[124] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.

[125] Wenbo Xu, Zain A. H. Hammadeh, Alexander Kröller, Rolf Ernst, and Sophie Quinton. Improved deadline miss models for real-time systems using typical worst-case analysis. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 247–256, July 2015.

[126] Fengxiang Zhang and Alan Burns. Schedulability analysis for real-time systems with edf scheduling. *IEEE Trans. Comput.*, 58(9):1250–1258, September 2009.

[127] Qi Zhu, Yang Yang, Eelco Scholte, Marco Di Natale, and Alberto Sangiovanni-Vincentelli. Optimizing extensibility in hard real-time distributed systems. In *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 275–284, April 2009.

[128] Dirk Ziegenbein, Arne Hamann, and Eckart Mayer-John. A suitable interface between function development and real-time systems integration, 2017. ESWEEK - Tutorial Slides.